

.....

```
EEEEEEEEEE RRRRRRRR FFFFFFFFFF
EEEEEEEEEE RRRRRRRR FFFFFFFFFF
EE          RR      RR FF
EE          RR      RR FF
EE          RR      RR FF
EE          RR      RR FF
EEEEEEEEEE RRRRRRRR FFFFFFFF
EEEEEEEEEE RRRRRRRR FFFFFFFF
EE          RR      RR FF
EE          RR      RR FF
EE          RR      RR FF
EE          RR      RR FF
EEEEEEEEEE RR      RR FF
EEEEEEEEEE RR      RR FF
```

....
....
....
....

```
LL          IIIIII SSSSSSSS
LL          IIIIII SSSSSSSS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SSSSSS
LL          II      SSSSSS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SS
LL          IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS
```



```
1 0001 0 MODULE ERF (XTITLE 'Errorlog Report Formatter'
2 0002 0      MAIN = ERF,
3 0003 0      IDENT = 'V04-000' ) =
4 0004 0 ! The version number above must be changed in two places.
5 0005 0 ! Search for 'IDENT ='.
6 0006 1 BEGIN
7 0007 1
8 0008 1
9 0009 1 *****
10 0010 1 *
11 0011 1 *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
12 0012 1 *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
13 0013 1 *  ALL RIGHTS RESERVED.
14 0014 1 *
15 0015 1 *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
16 0016 1 *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
17 0017 1 *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
18 0018 1 *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
19 0019 1 *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
20 0020 1 *  TRANSFERRED.
21 0021 1 *
22 0022 1 *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
23 0023 1 *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
24 0024 1 *  CORPORATION.
25 0025 1 *
26 0026 1 *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
27 0027 1 *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
28 0028 1 *
29 0029 1 *
30 0030 1 *****
31 0031 1
32 0032 1 ++
33 0033 1 FACILITY: ERF, Errorlog Report Formatter
34 0034 1
35 0035 1 ABSTRACT:
36 0036 1     This is the main routine for ERF. All exit paths must return
37 0037 1     here to exit.
38 0038 1
39 0039 1
40 0040 1 ENVIRONMENT:
41 0041 1
42 0042 1     VAX/VMS operating system. unprivileged user mode,
43 0043 1
44 0044 1 AUTHOR: Elliott A. Drayton
45 0045 1
46 0046 1 Modified by:
47 0047 1
48 0048 1     V03-026 EAD0197      Elliott A. Drayton      23-Jul-1984
49 0049 1     Added code to check version numbers in VALIDATE_PACKET.
50 0050 1
51 0051 1     V03-025 EAD0193      Elliott A. Drayton      6-Jul-1984
52 0052 1     Made LSTLUN own and initialized it from syecom. Cleared
53 0053 1     MAILBOX_CHANNEL which is now in syecom.
54 0054 1
55 0055 1     V03-024 EAD0180      Elliott A. Drayton      26-Jun-1984
56 0056 1     Add workstation device class and have /LOG report any
57 0057 1     files that were created.
```


58	0058	1			
59	0059	1	V03-023	EAD0170 Elliott A. Drayton	4-May-1984
60	0060	1		Added context parameter to FILE_SCAN.	
61	0061	1			
62	0062	1	V03-022	EAD0140 Elliott A. Drayton	12-Apr-1984
63	0063	1		Removed reference to EMBETDEF.	
64	0064	1			
65	0065	1	V03-021	EAD0130 Elliott A. Drayton	9-Apr-1984
66	0066	1		Moved image_loader and its support routines in to ERFshr.	
67	0067	1			
68	0068	1	V03-020	EAD0119 Elliott A. Drayton	22-Mar-1984
69	0069	1		Add support for the UNKNOWN keyword.	
70	0070	1			
71	0071	1	V03-019	EAD0117 Elliott A. Drayton	21-Mar-1984
72	0072	1		Fixed translation of loadable image name.	
73	0073	1			
74	0074	1	V03-018	SAR0209 Sharon A. Reynolds	10-Mar-1984
75	0075	1		- Fixed a bug in write_msg so it checks the contents of	
76	0076	1		output_flag.	
77	0077	1		- Changed the call to write_msg so that the address of the	
78	0078	1		command line descriptor is passed.	
79	0079	1			
80	0080	1	V03-017	EAD0116 Elliott A. Drayton	9-Mar-1984
81	0081	1		Removed emb_buf and syecom_buf.	
82	0082	1			
83	0083	1		JMG0013 Joel M. Gringorten	8-Mar-1984
84	0084	1		Output command line at end of report if IO is directed	
85	0085	1		to a file.	
86	0086	1			
87	0087	1	V03-016	EAD0107 Elliott A. Drayton	29-Feb-1984
88	0088	1		Fixed problem in validate_packet which prevented unknown	
89	0089	1		error packets from being identified.	
90	0090	1			
91	0091	1	V03-015	EAD0104 Elliott A. Drayton	29-Feb-1984
92	0092	1		Major clean up.	
93	0093	1			
94	0094	1		SAR0189 Sharon A. Reynolds	13-Feb-1984
95	0095	1		- Added the ERF_NOTFOUND message test at EOF.	
96	0096	1		- Added summary specific test for summary update calls.	
97	0097	1		- Added two parameters to the Brief_c_dispatcher calls.	
98	0098	1			
99	0099	1		JMG0012 Joel M. Gringorten	7-Feb-1984
100	0100	1		Added support for /statistics qualifier.	
101	0101	1			
102	0102	1	V03-014	JMG0005 Joel M. Gringorten	29-Dec-1983
103	0103	1		Added support for /summary=histogram.	
104	0104	1		- Added histo output dispatch to erf_control.	
105	0105	1		- Added histo update dispatch to process_file.	
106	0106	1			
107	0107	1	V03-013	SAR0172 Sharon A. Reynolds	16-Nov-1983
108	0108	1		- Added 'logmscp' entry support.	
109	0109	1		- Updated the 'max_xxx_type' values for new devices.	
110	0110	1		- Completed the device tables.	
111	0111	1		- Fixed a bug with /include=mem/summary=mem, the summary	
112	0112	1		statistics were counted twice.	
113	0113	1			
114	0114	1	V03-012	SAR0165 Sharon A. Reynolds	14-Oct-1983


```

: 115 0115 1 |
: 116 0116 1 |
: 117 0117 1 |
: 118 0118 1 |
: 119 0119 1 |
: 120 0120 1 |
: 121 0121 1 |
: 122 0122 1 |
: 123 0123 1 |
: 124 0124 1 |
: 125 0125 1 |
: 126 0126 1 |
: 127 0127 1 |
: 128 0128 1 |
: 129 0129 1 |
: 130 0130 1 |
: 131 0131 1 |
: 132 0132 1 |
: 133 0133 1 |
: 134 0134 1 |
: 135 0135 1 |
: 136 0136 1 |
: 137 0137 1 |
: 138 0138 1 |
: 139 0139 1 |
: 140 0140 1 |
: 141 0141 1 |
: 142 0142 1 |
: 143 0143 1 |
: 144 0144 1 |
: 145 0145 1 |
: 146 0146 1 |
: 147 0147 1 |
: 148 0148 1 |
: 149 0149 1 |
: 150 0150 1 |
: 151 0151 1 |
: 152 0152 1 |
: 153 0153 1 |
: 154 0154 1 |
: 155 0155 1 |
: 156 0156 1 |
: 157 0157 1 |
: 158 0158 1 |
: 159 0159 1 |
: 160 0160 1 |
: 161 0161 1 |
: 162 0162 1 |--

- Removed the code that counts logmessage/logstatus
  entries to fix a bug. (/incl=logm/excl=DU).
- Fixed a bug with the report type and /SID qualifier.
- Changed erf_norep to erf_invreptyp with fatal severity.

V03-011 SAR0150 Sharon A. Reynolds 7-Oct-1983
Fixed a bug in the 'validate_packet' routine.

V03-010 SAR0138 Sharon A. Reynolds 20-Sep-1983
Fixed bug in code that determines whether to init commons.

V03-009 SAR0136 Sharon A. Reynolds 12-Sep-1983
Added code that removed the mscp info msg first part.

V03-008 SAR0128 Sharon A. Reynolds 7-Sep-1983
Added routine to initialize the qiocommon, opcode,
and modes commons. (Init_commons routine). Replaced
debugging error messages with permanent error messages.

V03-007 EAD0006 Elliott A. Drayton 23-Aug-1983
Added routines to open and parse text library records which
are used to build internal tables.

V03-006 SAR0062 Sharon A. Reynolds, 20-Jun-1983
Fixed bug with processor type in 'validate_packet'.

V03-005 SAR0031 Sharon A. Reynolds, 2-Jun-1983
Put in a permanent solution to syecom buffer address problem.
Removed some unnecessary code.

V03-004 SAR0023 Sharon A. Reynolds, 11-May-1983
Modified 'process_packet' and 'full_dispatcher' so that the
summary information will be updated. Also modified
'process_record' so that multiple part MSCP entries will
be output. Fixed a problem passing record length.
Put in a temporary solution to syecom problem.

V03-003 SAR0011 Sharon A. Reynolds, 11-Apr-1983
Added code to 'validate_packet' to set up the device
class and type fields to the appropriate emb fields,
as they are in different locations per entry type.

V03-002 SAR0010 Sharon A. Reynolds, 9-Apr-1983
Intialized status, status2, status3, and status4. Also
added code to ensure processing a 'device entry' before
examining device class and device type in 'validate packet'.
```



```
164 0163 1 REQUIRE 'SRC$:RECSELDEF.REQ'; ! Defines emb fields
165 0294 1 REQUIRE 'LIB$:PARSERDAT.R32'; ! Defines option_flag fields
166 0448 1 REQUIRE 'SRC$:ERFDEF.REQ';
167 0734 1
168 0735 1
169 0736 1 FORWARD ROUTINE
170 0737 1 Build_class_tables, ! Allocates and inits device class tables
171 0738 1 Erf, ! Top level routine
172 0739 1 Erf_control, ! Main control loop
173 0740 1 Full_dispatcher, ! Cases to correct EXEC_IMAGE call
174 0741 1 Get_library_text, ! Reads library module record and calls parser
175 0742 1 Handler, ! Condition handler
176 0743 1 Init_commons, ! Initialize fortran data commons
177 0744 1 Open_text_lib, ! Routine to open and init the text library
178 0745 1 Parse_text_record, ! Routine to compress and parse text record
179 0746 1 Parse_max_table_size,
180 0747 1 Parse_module_names,
181 0748 1 Parse_device_desc_record,
182 0749 1 Parse_max_min_table_record,
183 0750 1 Process_file, ! Reads ERRLOG.SYS & loops till EOF
184 0751 1 Process_packet, ! Cases on report type to dispatcher
185 0752 1 Validate_packet, ! Checks packet for CPU,ENTRY,CLASS&TYPE
186 0753 1 Write_binary, ! Write packet as read, no text translation
187 0754 1 Write_err_msg; ! Write error messages to output file
188 0755 1
189 0756 1 EXTERNAL ROUTINE
190 0757 1 Exec_image, ! Call loaded image with correct params.
191 0758 1 Device_type_entry,
192 0759 1 Get_vm, ! Allocates requested buffers
193 0760 1 Image_loader, ! Determines which image to load & loads
194 0761 1 Lbr$close,
195 0762 1 Lbr$get_record,
196 0763 1 Lbr$ini_control,
197 0764 1 Lbr$lookup_key,
198 0765 1 Lbr$open,
199 0766 1 Lbr$set_locate,
200 0767 1 Lib$cvdt_dtb, ! Convert decimal to binary
201 0768 1 Lib$extzv,
202 0769 1 Lib$file_scan,
203 0770 1 Log_filename, ! Signals filenames and error messages
204 0771 1 Map_image,
205 0772 1 Parse_command, ! Analyze command line
206 0773 1 Parse_output_files, ! Handles the opening of output files
207 0774 1 Record_selected, ! Determines if record should be processed
208 0775 1 Timrb, ! Runtime statistics package
209 0776 1 Timre,
210 0777 1 Unknown_dispatcher, ! Formats and outputs reports for unknown error packets
211 0778 1 Write_msg;
212 0779 1
213 0780 1 EXTERNAL
214 0781 1 Class_dir: REF VECTOR[WORD],
215 0782 1 EMB: $BBLOCK PSECT (EMB),
216 0783 1 Input_fab: $BBLOCK [],
217 0784 1 Input_rab: $BBLOCK [],
218 0785 1 Input_nam: $BBLOCK [],
219 0786 1 Input_xabfhc: $BBLOCK [],
220 0787 1 Lstlun_rab_address: REF $BBLOCK [],
```



```
.. 221 0788 1 Option_flag: REF $BBLOCK [],
.. 222 0789 1 Output_fab: $BBLOCK [],
.. 223 0790 1 Output_nam: REF $BBLOCK [],
.. 224 0791 1 Output_rab: $BBLOCK [],
.. 225 0792 1 Parser_data: REF $BBLOCK [],
.. 226 0793 1 Parser_table: REF $BBLOCK [],
.. 227 0794 1 Related_nam: $BBLOCK [],
.. 228 0795 1 Rejected_fab: $BBLOCK [],
.. 229 0796 1 Rejected_nam: $BBLOCK [],
.. 230 0797 1 Rejected_rab: $BBLOCK [],
.. 231 0798 1 Summary_flag: REF $BBLOCK [],
.. 232 0799 1 Syecom: $BBLOCK PSÉCT (SYECOM),
.. 233 0800 1 Sys$output_rab_address: REF $BBLOCK,
.. 234 0801 1 Worst_error: $BBLOCK [LONG],
.. 235 0802 1 Lnm$file_dev_desc,
.. 236 0803 1 Bus_image: REF BLOCKVECTOR[2] FIELD (desc_fields),
.. 237 0804 1 Bus_version: REF VECTOR[WORD],
.. 238 0805 1 Bus_xfer_addr: REF VECTOR[LONG], ! Address of bus xfer address table
.. 239 0806 1 Disk_image: REF BLOCKVECTOR[2] FIELD (desc_fields),
.. 240 0807 1 Disk_version: REF VECTOR[WORD], ! Address of version number of device dependent code
.. 241 0808 1 Disk_xfer_addr: REF VECTOR[LONG], ! Address of disk xfer address table
.. 242 0809 1 Lp_image: REF BLOCKVECTOR[2] FIELD (desc_fields),
.. 243 0810 1 Lp_version: REF VECTOR[WORD],
.. 244 0811 1 Lp_xfer_addr: REF VECTOR[LONG], ! Address of lp xfer address table
.. 245 0812 1 Max_misc_type: BYTE,
.. 246 0813 1 Max_lp_type: BYTE,
.. 247 0814 1 Packet_processor_xfer_addr: REF VECTOR[LONG], ! Address of realtime xfer address table
.. 248 0815 1 Packet_processor_image: REF BLOCKVECTOR[2] FIELD (desc_fields),
.. 249 0816 1 Packet_processor_version: REF VECTOR[WORD],
.. 250 0817 1 Realtime_image: REF BLOCKVECTOR[2] FIELD (desc_fields),
.. 251 0818 1 Realtime_version: REF VECTOR[WORD],
.. 252 0819 1 Realtime_xfer_addr: REF VECTOR[LONG], ! Address of realtime xfer address table
.. 253 0820 1 Scm_image: REF BLOCKVECTOR[2] FIELD (desc_fields),
.. 254 0821 1 Scm_version: REF VECTOR[WORD],
.. 255 0822 1 Scm_xfer_addr: REF VECTOR[LONG], ! Address of scm xfer address table
.. 256 0823 1 Tape_image: REF BLOCKVECTOR[2] FIELD (desc_fields),
.. 257 0824 1 Tape_version: REF VECTOR[WORD],
.. 258 0825 1 Tape_xfer_addr: REF VECTOR[LONG], ! Address of tape xfer address table
.. 259 0826 1 Translate_entry_table: REF VECTOR[WORD],
.. 260 0827 1 Workstation_image: REF BLOCKVECTOR[2] FIELD (desc_fields),
.. 261 0828 1 Workstation_version: REF VECTOR[WORD],
.. 262 0829 1 Workstation_xfer_addr: REF VECTOR[LONG],
.. 263 0830 1
.. 264 0831 1
.. 265 0832 1 LITERAL
.. 266 0833 1 Erf$facility = 8, ! Facility code for ERF
.. 267 0834 1 Word_size = 2, ! Number of bytes in a word
.. 268 0835 1 Longword = 4, ! Number of bytes in a longword
.. 269 0836 1 Descriptor_length = 8; ! Number of bytes in a string descriptor
.. 270 0837 1
.. 271 0838 1 Own
.. 272 0839 1 Lstlun: LONG;
.. 273 0840 1
.. 274 0841 1 EXTERNAL LITERAL
.. 275 0842 1 Erf_badevtyp,
.. 276 0843 1 Erf_badevval,
.. 277 0844 1 Erf_badmodnam,
```

ERF
V04-000

Errorlog Report Formatter

J 2
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 6
(2)

```
: 278      0845 1 Erf_clstblerr,  
: 279      0846 1 Erf_cvterr,  
: 280      0847 1 Erf_herald,  
: 281      0848 1 Erf_input,  
: 282      0849 1 Erf_loaderr,  
: 283      0850 1 Erf_notfound,  
: 284      0851 1 Erf_invreptyp,  
: 285      0852 1 Erf_toomancs,  
: 286      0853 1 Erf_total,  
: 287      0854 1 Erf_unkentry,  
: 288      0855 1 Erf_unkclass;  
: 289      0856 1  
: 290      0857 1  
: 291      0858 1 SD('ERFLIB'); ! Defines string descriptors
```



```
.. 293      0859 1 GLOBAL
.. 294      0860 1   Bus_devices:
.. 295      0861 1   Desc_table_address:
.. 296      0862 1   Dev_addr_ptr:
.. 297      0863 1   Dev_class_ptr:
.. 298      0864 1   Disk_devices:
.. 299      0865 1   Function,
.. 300      0866 1   Herald:
.. 301      0867 1   Initd_commons,
.. 302      0868 1   Input_desc:
.. 303      0869 1   Input_file_count,
.. 304      0870 1   Item_count,
.. 305      0871 1   Library_func:
.. 306      0872 1   Library_index,
.. 307      0873 1   Library_type:
.. 308      0874 1   Lp_devices:
.. 309      0875 1   Max_bus_type:
.. 310      0876 1   Max_classes:
.. 311      0877 1   Max_cpu_types:
.. 312      0878 1   Max_disk_type:
.. 313      0879 1   Max_range_table_addr:
.. 314      0880 1   Max_realtime_type:
.. 315      0881 1   Max_scom_type:
.. 316      0882 1   Max_tape_type:
.. 317      0883 1   Max_Workstation_type:
.. 318      0884 1   Min_modules_desc:
.. 319      0885 1   Max_modules_desc:
.. 320      0886 1   Min_max_table_sizes:
.. 321      0887 1   Min_range_table_addr:
.. 322      0888 1   Module_name_desc,
.. 323      0889 1   Packet_processor_devices:
.. 324      0890 1   Processor_type_table:
.. 325      0891 1   Realtime_devices:
.. 326      0892 1   Record_desc:
.. 327      0893 1   Scom_devices:
.. 328      0894 1   Selected,
.. 329      0895 1   Summary_dispatcher_addr,
.. 330      0896 1   Tape_devices:
.. 331      0897 1   Table_address:
.. 332      0898 1   Table_length:
.. 333      0899 1   Text_rfa:
.. 334      0900 1   Token_desc:
.. 335      0901 1   Total_selected,
.. 336      0902 1   Total_rejected,
.. 337      0903 1   Unknown_entry:
.. 338      0904 1   Workstation_devices:
.. 339      0905 1
.. 340      0906 1 Builtin
.. 341      0907 1   FFS ;
.. 342      0908 1

REF VECTOR[WORD]
REF BLOCKVECTOR[2] FIELD (desc_fields),
REF VECTOR[LONG],      ! Address device name pointers
REF VECTOR[WORD],
REF VECTOR[WORD],      ! Address of disk device name table (e.g. DB)
                        ! Function to be done
$BBLOCK [12],          ! Address of message vector
$BBLOCK [dsc$k_d_bln], ! Allocate dynmaic descriptor
                        ! Used as index count
INITIAL (Lbr$c_read),
INITIAL (lbr$c_typ_txt),
REF VECTOR[WORD],
BYTE,
BYTE,
WORD,
BYTE,
REF VECTOR [LONG],
BYTE,
BYTE,
BYTE,
BYTE,
REF BLOCKVECTOR[2] FIELD (desc_fields),
REF BLOCKVECTOR[2] FIELD (desc_fields),
REF VECTOR[WORD],
REF VECTOR[LONG],
REF VECTOR[WORD],
REF VECTOR [WORD],
REF VECTOR[WORD],
$BBLOCK [dsc$k_d_bln],
REF VECTOR[WORD],
                        ! Count of records selected for processing
                        ! Transfer address of summary dispatcher
REF VECTOR[WORD],
REF VECTOR[WORD],
BYTE,
VECTOR [2],
$BBLOCK [DSC$K_D_BLN],
INITIAL (false),
REF VECTOR[WORD];
```

ERF
V04-000

Errorlog Report Formatter

L 2
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 8
(4)

```

344 0909 1 Routine ERF =
345 0910 2 BEGIN
346 0911 2 ++
347 0912 2 Functional description
348 0913 2
349 0914 2 This is the top level routine for the ERF facility.
350 0915 2 It calls the main control loop. Any errors encountered
351 0916 2 will be passed back to this routine.
352 0917 2
353 0918 2 Calling sequence
354 0919 2
355 0920 2 ERF () from the command language interpreter
356 0921 2
357 0922 2 Input parameters
358 0923 2
359 0924 2 AP = Address of argument list passed from CLI
360 0925 2
361 0926 2 Output parameters
362 0927 2
363 0928 2 None
364 0929 2
365 0930 2 Routine value
366 0931 2
367 0932 2 Worst error is returned.
368 0933 2
369 0934 2 ----
370 0935 2
371 0936 2 LOCAL
372 0937 2 channel,
373 0938 2 status;
374 0939 2
375 0940 2
376 0941 2 SET UP HANDLERS ---
377 0942 2 Declare condition handler to record severest
378 0943 2 error message issued, to be returned on exit of image.
379 0944 2
380 0945 2
381 0946 2 ENABLE handler;
382 0947 2
383 0948 2
384 0949 2
385 0950 2 CALL MAIN CONTROL ---
386 0951 2 Invoke the main subroutine. If any errors are encountered they
387 0952 2 will be returned immediately, if fatal, or saved in WORST_ERROR
388 0953 2 for exit processing.
389 0954 2
390 0955 2
391 0956 2 Worst_error = Erf_control() ;
392 0957 2
393 0958 2
394 0959 2
395 0960 2 RETURN TO USER ---
396 0961 2 Return to the user. Variable WORST_ERROR is maintained by
397 0962 2 the error handler (see routine HANDLER). If no messages have
398 0963 2 been signaled then the initial value of WORST_ERROR, $$$_NORMAL,
399 0964 2 will be returned.
400 0965 2
```


ERF
V04-000

Errorlog Report Formatter

M 2
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 9
(4)

```
: 401
: 402
: 403
: 404

0966 2
0967 2 Return .worst_error;
0968 2
0969 1 END;
```

! Return contents of WORST_ERROR

```
.TITLE ERF Errorlog Report Formatter
.IDENT \V04-000\

.PSECT $PLIT,NOWRT,NOEXE, PIC,2

42 49 4C 46 52 45 00000 P.AAB: .ASCII \ERFLIB\
00006 .BLKB 2
00000006 00008 P.AAA: .LONG 6
00000000 0000C .ADDRESS P.AAB

.PSECT $OWNS,NOEXE, PIC,2

00000 LSTLUN: .BLKB 4

.PSECT $GLOBAL$,NOEXE, PIC,2

00000 BUS_DEVICES::
.BKLB 4
00004 DESC_TABLE_ADDRESS::
.BKLB 4
00008 DEV_ADDR_PTR::
.BKLB 4
0000C DEV_CLASS_PTR::
.BKLB 4
00010 DISK_DEVICES::
.BKLB 4
00014 FUNCTION::
.BKLB 4
00018 HERALD:: .BLKB 12
00024 INITED_COMMONS::
.BKLB 4
00028 INPUT_DESC::
.BKLB 8
00030 INPUT_FILE_COUNT::
.BKLB 4
00034 ITEM_COUNT::
.BKLB 4
00000001 00038 LIBRARY_FUNC::
.LONG 1
0003C LIBRARY_INDEX::
.BKLB 4
00000004 00040 LIBRARY_TYPE::
.LONG 4
00044 LP_DEVICES::
.BKLB 4
00048 MAX_BUS_TYPE::
.BKLB 1
00049 MAX_CLASSES::
.BKLB 1
0004A MAX_CPU_TYPES::
.BKLB 2
```

N 2
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

```
0004C MAX_DISK_TYPE::
      .BLKB 1
0004D      .BLKB 3
00050 MAX_RANGE_TABLE_ADDR::
      .BLKB 4
00054 MAX_REALTIME_TYPE::
      .BLKB 1
00055 MAX_SCOM_TYPE::
      .BLKB 1
00056 MAX_TAPE_TYPE::
      .BLKB 1
00057 MAX_WORKSTATION_TYPE::
      .BLKB 1
00058 MIN_MODULES_DESC::
      .BLKB 4
0005C MAX_MODULES_DESC::
      .BLKB 4
00060 MIN_MAX_TABLE_SIZES::
      .BLKB 4
00064 MIN_RANGE_TABLE_ADDR::
      .BLKB 4
00068 MODULE_NAME_DESC::
      .BLKB 4
0006C PACKET_PROCESSOR_DEVICES::
      .BLKB 4
00070 PROCESSOR_TYPE_TABLE::
      .BLKB 4
00074 REALTIME_DEVICES::
      .BLKB 4
00078 RECORD_DESC::
      .BLKB 8
00080 SCOM_DEVICES::
      .BLKB 4
00084 $SELECTED::
      .BLKB 4
00088 SUMMARY_DISPATCHER_ADDR::
      .BLKB 4
0008C TAPE_DEVICES::
      .BLKB 4
00090 TABLE_ADDRESS::
      .BLKB 4
00094 TABLE_LENGTH::
      .BLKB 1
00095      .BLKB 3
00098 TEXT_RFA::
      .BLKB 8
000A0 TOKEN_DESC::
      .BLKB 8
000A8 TOTAL_SELECTED::
      .BLKB 4
000AC TOTAL_REJECTED::
      .BLKB 4
00000000 000B0 UNKNOWN_ENTRY::
      .LONG 0
000B4 WORKSTATION_DEVICES::
      .BLKB 4
```



```
ERFLIB_DESC== P.AAA
.EXTRN EXEC_IMAGE, DEVICE_TYPE_ENTRY
.EXTRN GET_VM, IMAGE_LOADER
.EXTRN LBR$CLOSE, LBR$GET_RECORD
.EXTRN LBR$INI CONTROL
.EXTRN LBR$LOOKUP_KEY, LBR$OPEN
.EXTRN LBR$SET LOCATE, LIB$CVT DTB
.EXTRN LIB$EXTZV, LIB$FILE_SCAN
.EXTRN LOG_FILENAME, MAP_IMAGE
.EXTRN PARSE_COMMAND, PARSE_OUTPUT_FILES
.EXTRN RECORD_SELECTED
.EXTRN TIMRB, TIMRE, UNKNOWN_DISPATCHER
.EXTRN WRITE_MSG, CLASS_DIR
.EXTRN EMB, INPUT_FAB, INPUT_RAB
.EXTRN INPUT_NAM, INPUT_XABFRC
.EXTRN LSTLUN_RAB_ADDRESS
.EXTRN OPTION_FLAG, OUTPUT_FAB
.EXTRN OUTPUT_NAM, OUTPUT_RAB
.EXTRN PARSER_DATA, PARSER_TABLE
.EXTRN RELATED_NAM, REJECTED_FAB
.EXTRN REJECTED_NAM, REJECTED_RAB
.EXTRN SUMMARY_FLAG, SYECOM
.EXTRN SYSSOUTPUT_RAB_ADDRESS
.EXTRN WORST_ERROR, LNM$FILE_DEV_DESC
.EXTRN BUS_IMAGE, BUS_VERSION
.EXTRN BUS_XFER_ADDR, DISK_IMAGE
.EXTRN DISK_VERSION, DISK_XFER_ADDR
.EXTRN LP_IMAGE, LP_VERSION
.EXTRN LP_XFER_ADDR, MAX_MISC_TYPE
.EXTRN MAX_LP_TYPE, PACKET_PROCESSOR_XFER_ADDR
.EXTRN PACKET_PROCESSOR_IMAGE
.EXTRN PACKET_PROCESSOR_VERSION
.EXTRN REALTIME_IMAGE, REALTIME_VERSION
.EXTRN REALTIME_XFER_ADDR
.EXTRN SCOM_IMAGE, SCOM_VERSION
.EXTRN SCOM_XFER_ADDR, TAPE_IMAGE
.EXTRN TAPE_VERSION, TAPE_XFER_ADDR
.EXTRN TRANSLATE_ENTRY_TABLE
.EXTRN WORKSTATION_IMAGE
.EXTRN WORKSTATION_VERSION
.EXTRN WORKSTATION_XFER_ADDR
.EXTRN ERF_BADEVTYP, ERF_BADEVVAL
.EXTRN ERF_BADMODNAM, ERF_CLSTBLERR
.EXTRN ERF_CVTERR, ERF_HERALD
.EXTRN ERF_INPUT, ERF_LOADERR
.EXTRN ERF_NOTFOUND, ERF_INVREPTYP
.EXTRN ERF_TOOMANCLS, ERF_TOTAL
.EXTRN ERF_UNKENTRY, ERF_ONKCLASS
```

```
.PSECT $CODE, NOWRT, PIC.2
```

00000000V	6D	0010	0000 00000	ERF:	.WORD	Save nothing	:	0909
00000000G	00		CF DE 00002		MOVAL	1\$, (FP)	:	0910
	00		00 FB 00007		CALLS	#0, ERF_CONTROL	:	0956
	00		50 D0 0000E		MOVL	R0, WORST_ERROR	:	
			04 00015		RET		:	0969
			0000 00016	1\$:	.WORD	Save nothing	:	0910

ERF
V04-000

Errorlog Report Formatter

C 3
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 12
(4)

00000000V 7E 00

04

7E D4 00018
5E DD 0001A
AC 7D 0001C
03 FB 00020
04 00027

CLRL -(SP)
PUSHL SP
MOVQ 4(AP), -(SP)
CALLS #3, HANDLER
RET

:
:
:
:
:

; Routine Size: 40 bytes, Routine Base: \$CODE + 0000


```
: 406      0970 1 Routine ERF_CONTROL =                ! Main control loop for ERF
: 407      0971 2 BEGIN
: 408      0972 2
: 409      0973 2 |++
: 410      0974 2 | Functional description
: 411      0975 2 |
: 412      0976 2 |     This is the control routine for the ERF facility.
: 413      0977 2 |     Any errors encountered will be passed back to this routine.
: 414      0978 2 |
: 415      0979 2 | Calling sequence
: 416      0980 2 |
: 417      0981 2 |     ERF_CONTROL ()
: 418      0982 2 |
: 419      0983 2 | Input parameters
: 420      0984 2 |
: 421      0985 2 |
: 422      0986 2 | Output parameters
: 423      0987 2 |
: 424      0988 2 |     None
: 425      0989 2 |
: 426      0990 2 | Routine value
: 427      0991 2 |
: 428      0992 2 |     Worst error is returned.
: 429      0993 2 |
: 430      0994 2 | ----
: 431      0995 2 |
: 432      0996 2 | LOCAL
: 433      0997 2 |     Field_size:                BYTE INITIAL (6),
: 434      0998 2 |     Scan_context:              INITIAL (0),
: 435      0999 2 |     Start_position:            INITIAL (0),
: 436      1000 2 |     Status,                                ! Returned status
: 437      1001 2 |     Summary_index,                    ! Index for summary type
: 438      1002 2 |     Text_lib_name,
: 439      1003 2 |     Out_file;
: 440      1004 2 |
: 441      1005 2 |
: 442      1006 2 |
: 443      1007 2 | ! Unconditional call to the runtime statistics package initialization routine.
: 444      1008 2 |
: 445      1009 2 | TIMRB();
: 446      1010 2 |
: 447      1011 2 |
: 448      1012 2 | $INIT_DYNDESC (input_desc);
: 449      1013 2 |
: 450      1014 2 |
: 451      1015 2 | ! Initialize the number of lines output.
: 452      1016 2 |
: 453      1017 2 | Syecom[sye$b_lines] = 1 ;
: 454      1018 2 |
```

```

456 1019 2
457 1020 2
458 1021 2 OPEN TEXT LIBRARY
459 1022 2 Call LBR interface routines to prepare the library which
460 1023 2 will provide text for output reports.
461 1024 2
462 1025 2 CALL_FUNCTION ( Open_text_lib ( ) );
463 1026 2
464 1027 2
465 1028 2
466 1029 2 PROCESS COMMAND LINE
467 1030 2 Call CLI interface routines to parse command line and setup
468 1031 2 internal tables for further processing.
469 1032 2
470 1033 2 CALL_FUNCTION ( Parse_command ( ) );
471 1034 2
472 1035 2
473 1036 2 If /SUMMARY then load the ERFSUMM.EXE. If ERFSUMM.EXE not found clear
474 1037 2 summary flag and continue.
475 1038 2
476 1039 2
477 1040 2 If .option_flag [opt$v_summary_qual] then
478 1041 2 Begin
479 1042 2 Status = Map_image( AD ('SYS$SYSTEM:ERFSUMM.EXE'), Summary_dispatcher_addr) ;
480 1043 2 If NOT .status then option_flag [opt$v_summary_qual] = 0;
481 1044 2 End;
482 1045 2
483 1046 2
484 1047 2 Syecom[sysel_mailbox_channel] = 0;
485 1048 2
486 1049 2
487 1050 2 While file names exist in the command line go process the file.
488 1051 2
489 1052 2
490 1053 2 While GET_VALUE ('FILE_SPECS', input_desc ) do
491 1054 2 Begin
492 1055 2 input_fab [fab$b_fns] = .input_desc [dsc$w_length];
493 1056 2 input_fab [fab$l_fna] = .input_desc [dsc$a_pointer];
494 1057 2 input_fab [fab$l_ctx] = msg$_searchfail;
495 1058 2 LIB$FILE_SCAN (
496 1059 2 Input_fab,
497 1060 2 Process_file,
498 1061 2 Log_filename,
499 1062 2 Scan_context );
500 1063 2 End;
501 1064 2
502 1065 2
503 1066 2
504 1067 2 If /SUMMARY then output summary report type requested.
505 1068 2
506 1069 2 If .option_flag[opt$v_summary_qual]
507 1070 2 Then
508 1071 2 Begin
509 1072 2
510 1073 2 Until .start_position GTR 6 do
511 1074 2 Begin
512 1075 2
```



```

513      1076 4      FFS (start_position,field_size,
514      1077 4          .summary_flag,summary_index) ;
515      1078 4
516      1079 4      Case .summary_index from 0 to 5 of
517      1080 4          Set
518      1081 4
519      1082 4          [0]:
520      1083 4              Function = all_summ_out;      ! Initialize function to output
521      1084 4                  ! all possible summary information
522      1085 4
523      1086 4          [1]:
524      1087 4              Function = dev_summ_out;      ! Initialize function to output
525      1088 4                  ! device summary information only
526      1089 4
527      1090 4          [2]:
528      1091 4              Function = entry_summ_out;    ! Initialize function to output
529      1092 4                  ! entry summary information only
530      1093 4
531      1094 4          [3]:
532      1095 4              Function = memory_summ_out;    ! Initialize function to output
533      1096 4                  ! memory summary information only
534      1097 4
535      1098 4          [4]:
536      1099 4              Function = volume_summ_out;    ! Initialize function to output
537      1100 4                  ! volume summary information only
538      1101 4
539      1102 4          [5]:
540      1103 4              Function = histo_summ_out;     ! Initialize function to output
541      1104 4                  ! histogram summary information only
542      1105 4
543      1106 4          [OUTRANGE]:
544      1107 4              EXITLOOP;
545      1108 4
546      1109 4          TES;
547      1110 4
548      1111 4          Exec_image ( Summary_dispatcher_addr, lstlun, Function) ;
549      1112 4          Start_position = .summary_index + 1 ;
550      1113 3          End ;
551      1114 2      End;
552      1115 2
553      1116 2
554      1117 2      ! LOG MESSAGE
555      1118 2      ! If /LOG requested and more then one input file
556      1119 2      ! was processed then print total number of files processed,
557      1120 2      ! total records selected and total records rejected.
558      1121 2      !
559      1122 2
560      1123 2      If .option_flag[opt$v_log_qual] and .input_file_count gtru 1 then
561      1124 2          signal (erf_total, 3, .total_selected, .total_rejected, .input_file_count);
562      1125 2
563      1126 2
564      1127 2      !
565      1128 2      ! If /STATISTICS was specified then call the runtime statistics display routine.
566      1129 2      !
567      1130 2      If .option_flag[opt$v_statistics_qual] then TIMRE(lstlun) ;
568      1131 2
569      1132 2
```

```
570 1133 2 !
571 1134 2 ! Write original command line, if /REJECTED was not specified.
572 1135 2 !
573 1136 2 If ( NOT .option_flag[opt$rejected_qual] ) then
574 1137 2   write_msg ( parser_table[erl$cmd_line], 1 );
575 1138 2
576 1139 2
577 1140 2 !
578 1141 2 ! CLOSE OUTPUT FILES
579 1142 2 !
580 1143 2
581 1144 2 If .syecom [syel_forms] OR
582 1145 2   .option_flag [opt$binary_qual]
583 1146 2 then
584 1147 2   BEGIN
585 1148 2     If .option_flag[opt$log_qual]
586 1149 2     then
587 1150 2       BEGIN
588 1151 2         Local desc : VECTOR [2,long];
589 1152 2         Desc[0] = .output_nam [nam$b_rsl];
590 1153 2         Desc[1] = .output_nam [nam$l_rsa];
591 1154 2         Signal (msg$created, 1, desc);
592 1155 2       END;
593 1156 2     END;
594 1157 2
595 1158 2
596 1159 2 Output_fab [fab$l_ctx] = msg$closeout;      ! Assign error messages
597 1160 2 Rejected_fab [fab$l_ctx] = msg$closeout;    ! Assign error messages
598 1161 2
599 1162 2 If .option_flag [opt$output_qual] AND
600 1163 2   .option_flag [opt$binary_qual]
601 1164 2 then
602 1165 2   CALL_FUNCTION ( $close (fab = output_fab, err = log_filename) );
603 1166 2
604 1167 2
605 1168 2 If .option_flag[opt$rejected_qual]
606 1169 2 then
607 1170 2   BEGIN
608 1171 2     If .option_flag[opt$log_qual]
609 1172 2     then
610 1173 2       BEGIN
611 1174 2         Local desc : VECTOR [2,long];
612 1175 2         Desc[0] = .rejected_nam [nam$b_rsl];
613 1176 2         Desc[1] = .rejected_nam [nam$l_rsa];
614 1177 2         Signal (msg$created, 1, desc);
615 1178 2       END;
616 1179 2     CALL_FUNCTION ( $close ( fab = rejected_fab, err = log_filename) );
617 1180 2   END;
618 1181 2
619 1182 2 Return .worst_error
620 1183 2
621 1184 1 End;
```

.PSECT \$PLIT,NOWRT,NOEXE, PIC,2


```
53 46 52 45 3A 4D 45 54 53 59 53 24 53 59 53 00010 P.AAD: .ASCII \SYS$SYSTEM:ERFSUMM.EXE\<0><0>
00 00 53 43 45 50 53 5F 45 4C 49 46 0001F P.AAC: .LONG 22
00000000' 00028 P.AAF: .ADDRESS P.AAD
00000000' 0002C P.AAF: .ASCII \FILE_SPECS\<0><0>
0000000A' 00030 P.AAE: .LONG 10
00000000' 0003C P.AAE: .ADDRESS P.AAF
00000000' 00040
      .EXTRN CLISGET_VALUE, SYS$CLOSE
      .PSECT $CODE,NOWRT, PIC,2

OFFC 00000 ERF_CONTROL:
      .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11
5B 00000000' 00 9E 00002 MOVAB LSTLUN, R11
5A 00000000G 00 9E 00009 MOVAB LIB$SIGNAL, R10
59 00000000G 00 9E 00010 MOVAB LOG_FILENAME, R9
58 00000000G 00 9E 00017 MOVAB SYECOM+12, R8
57 00000000G 00 9E 0001E MOVAB INPUT_FAB+52, R7
56 00000000G 00 9E 00025 MOVAB OPTION_FLAG, R6
55 00000000' 00 9E 0002C MOVAB FUNCTION, R5
5E 08 C2 00033 SUBL2 #8, SP
54 06 90 00036 MOVAB #6, FIELD_SIZE
      7E D4 00039 CLRL SCAN_CONTEXT
      53 D4 0003B CLRL START_POSITION
00000000G 00 00 FB 0003D CALLS #0, TIMRB
      14 A5 020E0000 8F D0 00044 MOVL #34471936, INPUT_DESC
      18 A5 D4 0004C CLRL INPUT_DESC+4
      68 01 90 0004F MOVAB #1, SYECOM+12
00000000V 00 00 FB 00052 CALLS #0, OPEN_TEXT_LIB
      07 50 E9 00059 BLBC STATUS, 1$
00000000G 00 00 FB 0005C CALLS #0, PARSE_COMMAND
      01 50 E8 00063 1$: BLBS STATUS, 2$
      04 00066 RET
      50 66 D0 00067 2$: MOVL OPTION_FLAG, R0
1B 60 0E E1 0006A BBC #14, (R0), 3$
      74 A5 9F 0006E PUSHAB SUMMARY_DISPATCHER_ADDR
      00000000' 00 9F 00071 PUSHAB P.AAC
00000000G 00 02 FB 00077 CALLS #2, MAP_IMAGE
      08 50 E8 0007E BLBS STATUS, 3$
      50 66 D0 00081 MOVL OPTION_FLAG, R0
      01 A0 8F 8A 00084 BICB2 #64, 1(R0)
      13 A8 D4 00089 3$: CLRL SYECOM+31
      14 A5 9F 0008C 4$: PUSHAB INPUT_DESC
00000000G 00 00 9F 0008F PUSHAB P.AAE
      27 50 E9 00095 CALLS #2, CLISGET_VALUE
      67 14 A5 90 0009F BLBC R0, 5$
      F8 A7 18 A5 D0 000A3 MOVAB INPUT_DESC, INPUT_FAB+52
      E4 A7 0008123A 8F D0 000A8 MOVL INPUT_DESC+4, INPUT_FAB+44
      4200 8F BB 000B0 MOVL #528954, INPUT_FAB+24
      00000000V 00 00 9F 000B4 PUSHR #^M<R9, SP>
      CC A7 9F 000BA PUSHAB PROCESS_FILE
00000000G 00 04 FB 000BD PUSHAB INPUT_FAB
      50 C6 11 000C4 CALLS #4, LIB$FILE_SCAN
      60 66 D0 000C6 5$: BRB 4$
      0E E1 000C9 MOVL OPTION_FLAG, R0
      BBC #14, (R0), 15$
```

52	60	00	53	D1	000CD	6\$:	CMPL	START_POSITION, #6	1073
		50	4E	14	000D0		BGTR	15\$	
		54	00	D0	000D2		MOVL	SUMMARY_FLAG, R0	1077
			53	EA	000D9		FFS	START_POSITION, FIELD_SIZE, (R0), -	1076
								SUMMARY_INDEX	
001D	05	00	52	CF	000DE		CASEL	SUMMARY_INDEX, #0, #5	1079
	0018	0013	000E		000E2	7\$:	.WORD	8\$-7\$,-	
		0027	0022		000EA			9\$-7\$,-	
								10\$-7\$,-	
								11\$-7\$,-	
								12\$-7\$,-	
								13\$-7\$	
			30	11	000EE		BRB	15\$	1107
		65	01	D0	000F0	8\$:	MOVL	#1, FUNCTION	1083
			17	11	000F3		BRB	14\$	
		65	02	D0	000F5	9\$:	MOVL	#2, FUNCTION	1087
			12	11	000F8		BRB	14\$	
		65	04	D0	000FA	10\$:	MOVL	#4, FUNCTION	1091
			0D	11	000FD		BRB	14\$	
		65	06	D0	000FF	11\$:	MOVL	#6, FUNCTION	1095
			08	11	00102		BRB	14\$	
		65	07	D0	00104	12\$:	MOVL	#7, FUNCTION	1099
			03	11	00107		BRB	14\$	
		65	09	D0	00109	13\$:	MOVL	#9, FUNCTION	1103
			55	DD	0010C	14\$:	PUSHL	R5	1111
			5B	DD	0010E		PUSHL	R11	
			74	A5	9F 00110		PUSHAB	SUMMARY_DISPATCHER_ADDR	
		00000000G	00	03	FB 00113		CALLS	#3, EXEC_IMAGE	
			53	01	A2 9E 0011A		MOVAB	1(R2), START_POSITION	1112
					AD 11 0011E		BRB	6\$	1073
		50	66	D0	00120	15\$:	MOVL	OPTION_FLAG, R0	1123
			60	95	00123		TSTB	(R0)	
			19	18	00125		BGEQ	16\$	
		01	A5	D1	00127		CMPL	INPUT_FILE_COUNT, #1	
			13	1B	0012B		BLEQU	16\$	
			A5	DD	0012D		PUSHL	INPUT_FILE_COUNT	1124
		7E	C5	7D	00130		MOVQ	TOTAL_SELECTED, -(SP)	
			03	DD	00135		PUSHL	#3	
		00000000G	8F	DD	00137		PUSHL	#ERF TOTAL	
			05	FB	0013D		CALLS	#5, [IB\$SIGNAL	
		6A	66	D0	00140	16\$:	MOVL	OPTION_FLAG, R0	1130
		50	A0	E9	00143		BLBC	2(R0), -17\$	
		09	5B	DD	00147		PUSHL	R11	
			01	FB	00149		CALLS	#1, TIMRE	
		00000000G	66	D0	00150	17\$:	MOVL	OPTION_FLAG, R0	1136
			0A	E0	00153		BBS	#10, (R0), 18\$	
OF		60	01	DD	00157		PUSHL	#1	1137
			00	DD	00159		PUSHL	PARSER_TABLE	
		00000000G	02	FB	0015F		CALLS	#2, WRITE_MSG	
			A8	E8	00166	18\$:	BLBS	SYECOM+4, -19\$	1144
		50	66	D0	0016A		MOVL	OPTION_FLAG, R0	1145
		60	01	E1	0016D		BBC	#1, (R0), 20\$	
26		50	66	D0	00171	19\$:	MOVL	OPTION_FLAG, R0	1148
			60	95	00174		TSTB	(R0)	
			1F	18	00176		BGEQ	20\$	
		50	00	D0	00178		MOVL	OUTPUT_NAM, R0	1152
04		AE	03	A0	9A 0017F		MOVZBL	3(R0), -DESC	

ERF
V04-000

Errorlog Report Formatter

J 3
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 19
(6)

08	AE	04	A0	D0	00184	MOVL	4(R0), DESC+4	:	1153
		04	AE	9F	00189	PUSHAB	DESC	:	1154
			01	DD	0018C	PUSHL	#1	:	
		00081073	8F	DD	0018E	PUSHL	#528499	:	
			03	FB	00194	CALLS	#3, LIB\$SIGNAL	:	
	00000000G	00	0008105A	8F	D0	00197	20\$: MOVL	#528474, OUTPUT_FAB+24	1159
	00000000G	00	0008105A	8F	D0	001A2	MOVL	#528474, REJECTED_FAB+24	1160
		50		66	D0	001AD	MOVL	OPTION_FLAG, R0	1162
		16	01	A0	E9	001B0	BLBC	1(R0), -21\$	
12		60		01	E1	001B4	BBC	#1, (R0), 21\$	1163
				59	DD	001B8	PUSHL	R9	1165
		00000000G	00	9F	001BA	PUSHAB	OUTPUT_FAB	:	
	00000000G	00		02	FB	001C0	CALLS	#2, SYS\$CLOSE	
		42		50	E9	001C7	BLBC	STATUS, 24\$	
		50		66	D0	001CA	21\$: MOVL	OPTION_FLAG, R0	1168
34		60		0A	E1	001CD	BBC	#10, (R0), 23\$	
			60	95	001D1	TSTB	(R0)	:	1171
			1E	18	001D3	BGEQ	22\$:	
04	AE	00000000G	00	9A	001D5	MOVZBL	REJECTED_NAM+3, DESC	:	1175
08	AE	00000000G	00	D0	001DD	MOVL	REJECTED_NAM+4, DESC+4	:	1176
		04	AE	9F	001E5	PUSHAB	DESC	:	1177
			01	DD	001E8	PUSHL	#1	:	
		00081073	8F	DD	001EA	PUSHL	#528499	:	
			03	FB	001F0	CALLS	#3, LIB\$SIGNAL	:	
			59	DD	001F3	22\$: PUSHL	R9	:	1179
		00000000G	00	9F	001F5	PUSHAB	REJECTED_FAB	:	
	00000000G	00		02	FB	001FB	CALLS	#2, SYS\$CLOSE	
		07		50	E9	00202	BLBC	STATUS, 24\$	
		50	00000000G	00	D0	00205	23\$: MOVL	WORST_ERROR, R0	1182
				04	0020C	24\$: RET		:	1184

; Routine Size: 525 bytes, Routine Base: \$CODE + 0028

; 622 1185 1

```

: 624      1186 1 Routine PROCESS_FILE (FAB) =
: 625      1187 1
: 626      1188 1 ++
: 627      1189 1 Functional description
: 628      1190 1
: 629      1191 1     This routine processes one input file. It is
: 630      1192 1     called as an action routine from LIB$FILE_SCAN.
: 631      1193 1
: 632      1194 1 Calling sequence
: 633      1195 1
: 634      1196 1     PROCESS_FILE (FAB)
: 635      1197 1
: 636      1198 1 Input parameters
: 637      1199 1
: 638      1200 1     FAB - address of input_fab.
: 639      1201 1
: 640      1202 1 Output parameters
: 641      1203 1
: 642      1204 1     None
: 643      1205 1
: 644      1206 1 Routine value
: 645      1207 1
: 646      1208 1     Worst error is returned.
: 647      1209 1
: 648      1210 1 ----
: 649      1211 1
: 650      1212 2 BEGIN
: 651      1213 2
: 652      1214 2 Map
: 653      1215 2     Fab:          ref $bblock;
: 654      1216 2
: 655      1217 2
: 656      1218 2 Local
: 657      1219 2     Status:      $BBLOCK [LONG],
: 658      1220 2     Desc:        VECTOR [2,LONG];      ! General purpose descriptor
: 659      1221 2
: 660      1222 2
: 661      1223 2 Own
: 662      1224 2     Class:        WORD,
: 663      1225 2     First_time:  INITIAL (TRUE),
: 664      1226 2     Type:         WORD,
: 665      1227 2     Xfer_addr:   LONG ;
: 666      1228 2
: 667      1229 2
: 668      1230 2
: 669      1231 2
: 670      1232 2
: 671      1233 2
: 672      1234 2
: 673      1235 2
: 674      1236 2
: 675      1237 2
: 676      1238 2
: 677      1239 2
: 678      1240 2
: 679      1241 2
: 680      1242 2
: 681      1243 2
: 682      1244 2
: 683      1245 2
: 684      1246 2
: 685      1247 2
: 686      1248 2
: 687      1249 2
: 688      1250 2
: 689      1251 2
: 690      1252 2
: 691      1253 2
: 692      1254 2
: 693      1255 2
: 694      1256 2
: 695      1257 2
: 696      1258 2
: 697      1259 2
: 698      1260 2
: 699      1261 2
: 700      1262 2
: 701      1263 2
: 702      1264 2
: 703      1265 2
: 704      1266 2
: 705      1267 2
: 706      1268 2
: 707      1269 2
: 708      1270 2
: 709      1271 2
: 710      1272 2
: 711      1273 2
: 712      1274 2
: 713      1275 2
: 714      1276 2
: 715      1277 2
: 716      1278 2
: 717      1279 2
: 718      1280 2
: 719      1281 2
: 720      1282 2
: 721      1283 2
: 722      1284 2
: 723      1285 2
: 724      1286 2
: 725      1287 2
: 726      1288 2
: 727      1289 2
: 728      1290 2
: 729      1291 2
: 730      1292 2
: 731      1293 2
: 732      1294 2
: 733      1295 2
: 734      1296 2
: 735      1297 2
: 736      1298 2
: 737      1299 2
: 738      1300 2
: 739      1301 2
: 740      1302 2
: 741      1303 2
: 742      1304 2
: 743      1305 2
: 744      1306 2
: 745      1307 2
: 746      1308 2
: 747      1309 2
: 748      1310 2
: 749      1311 2
: 750      1312 2
: 751      1313 2
: 752      1314 2
: 753      1315 2
: 754      1316 2
: 755      1317 2
: 756      1318 2
: 757      1319 2
: 758      1320 2
: 759      1321 2
: 760      1322 2
: 761      1323 2
: 762      1324 2
: 763      1325 2
: 764      1326 2
: 765      1327 2
: 766      1328 2
: 767      1329 2
: 768      1330 2
: 769      1331 2
: 770      1332 2
: 771      1333 2
: 772      1334 2
: 773      1335 2
: 774      1336 2
: 775      1337 2
: 776      1338 2
: 777      1339 2
: 778      1340 2
: 779      1341 2
: 780      1342 2
: 781      1343 2
: 782      1344 2
: 783      1345 2
: 784      1346 2
: 785      1347 2
: 786      1348 2
: 787      1349 2
: 788      1350 2
: 789      1351 2
: 790      1352 2
: 791      1353 2
: 792      1354 2
: 793      1355 2
: 794      1356 2
: 795      1357 2
: 796      1358 2
: 797      1359 2
: 798      1360 2
: 799      1361 2
: 800      1362 2
: 801      1363 2
: 802      1364 2
: 803      1365 2
: 804      1366 2
: 805      1367 2
: 806      1368 2
: 807      1369 2
: 808      1370 2
: 809      1371 2
: 810      1372 2
: 811      1373 2
: 812      1374 2
: 813      1375 2
: 814      1376 2
: 815      1377 2
: 816      1378 2
: 817      1379 2
: 818      1380 2
: 819      1381 2
: 820      1382 2
: 821      1383 2
: 822      1384 2
: 823      1385 2
: 824      1386 2
: 825      1387 2
: 826      1388 2
: 827      1389 2
: 828      1390 2
: 829      1391 2
: 830      1392 2
: 831      1393 2
: 832      1394 2
: 833      1395 2
: 834      1396 2
: 835      1397 2
: 836      1398 2
: 837      1399 2
: 838      1400 2
: 839      1401 2
: 840      1402 2
: 841      1403 2
: 842      1404 2
: 843      1405 2
: 844      1406 2
: 845      1407 2
: 846      1408 2
: 847      1409 2
: 848      1410 2
: 849      1411 2
: 850      1412 2
: 851      1413 2
: 852      1414 2
: 853      1415 2
: 854      1416 2
: 855      1417 2
: 856      1418 2
: 857      1419 2
: 858      1420 2
: 859      1421 2
: 860      1422 2
: 861      1423 2
: 862      1424 2
: 863      1425 2
: 864      1426 2
: 865      1427 2
: 866      1428 2
: 867      1429 2
: 868      1430 2
: 869      1431 2
: 870      1432 2
: 871      1433 2
: 872      1434 2
: 873      1435 2
: 874      1436 2
: 875      1437 2
: 876      1438 2
: 877      1439 2
: 878      1440 2
: 879      1441 2
: 880      1442 2
: 881      1443 2
: 882      1444 2
: 883      1445 2
: 884      1446 2
: 885      1447 2
: 886      1448 2
: 887      1449 2
: 888      1450 2
: 889      1451 2
: 890      1452 2
: 891      1453 2
: 892      1454 2
: 893      1455 2
: 894      1456 2
: 895      1457 2
: 896      1458 2
: 897      1459 2
: 898      1460 2
: 899      1461 2
: 900      1462 2
: 901      1463 2
: 902      1464 2
: 903      1465 2
: 904      1466 2
: 905      1467 2
: 906      1468 2
: 907      1469 2
: 908      1470 2
: 909      1471 2
: 910      1472 2
: 911      1473 2
: 912      1474 2
: 913      1475 2
: 914      1476 2
: 915      1477 2
: 916      1478 2
: 917      1479 2
: 918      1480 2
: 919      1481 2
: 920      1482 2
: 921      1483 2
: 922      1484 2
: 923      1485 2
: 924      1486 2
: 925      1487 2
: 926      1488 2
: 927      1489 2
: 928      1490 2
: 929      1491 2
: 930      1492 2
: 931      1493 2
: 932      1494 2
: 933      1495 2
: 934      1496 2
: 935      1497 2
: 936      1498 2
: 937      1499 2
: 938      1500 2
: 939      1501 2
: 940      1502 2
: 941      1503 2
: 942      1504 2
: 943      1505 2
: 944      1506 2
: 945      1507 2
: 946      1508 2
: 947      1509 2
: 948      1510 2
: 949      1511 2
: 950      1512 2
: 951      1513 2
: 952      1514 2
: 953      1515 2
: 954      1516 2
: 955      1517 2
: 956      1518 2
: 957      1519 2
: 958      1520 2
: 959      1521 2
: 960      1522 2
: 961      1523 2
: 962      1524 2
: 963      1525 2
: 964      1526 2
: 965      1527 2
: 966      1528 2
: 967      1529 2
: 968      1530 2
: 969      1531 2
: 970      1532 2
: 971      1533 2
: 972      1534 2
: 973      1535 2
: 974      1536 2
: 975      1537 2
: 976      1538 2
: 977      1539 2
: 978      1540 2
: 979      1541 2
: 980      1542 2
: 981      1543 2
: 982      1544 2
: 983      1545 2
: 984      1546 2
: 985      1547 2
: 986      1548 2
: 987      1549 2
: 988      1550 2
: 989      1551 2
: 990      1552 2
: 991      1553 2
: 992      1554 2
: 993      1555 2
: 994      1556 2
: 995      1557 2
: 996      1558 2
: 997      1559 2
: 998      1560 2
: 999      1561 2
: 1000     1562 2
: 1001     1563 2
: 1002     1564 2
: 1003     1565 2
: 1004     1566 2
: 1005     1567 2
: 1006     1568 2
: 1007     1569 2
: 1008     1570 2
: 1009     1571 2
: 1010     1572 2
: 1011     1573 2
: 1012     1574 2
: 1013     1575 2
: 1014     1576 2
: 1015     1577 2
: 1016     1578 2
: 1017     1579 2
: 1018     1580 2
: 1019     1581 2
: 1020     1582 2
: 1021     1583 2
: 1022     1584 2
: 1023     1585 2
: 1024     1586 2
: 1025     1587 2
: 1026     1588 2
: 1027     1589 2
: 1028     1590 2
: 1029     1591 2
: 1030     1592 2
: 1031     1593 2
: 1032     1594 2
: 1033     1595 2
: 1034     1596 2
: 1035     1597 2
: 1036     1598 2
: 1037     1599 2
: 1038     1600 2
: 1039     1601 2
: 1040     1602 2
: 1041     1603 2
: 1042     1604 2
: 1043     1605 2
: 1044     1606 2
: 1045     1607 2
: 1046     1608 2
: 1047     1609 2
: 1048     1610 2
: 1049     1611 2
: 1050     1612 2
: 1051     1613 2
: 1052     1614 2
: 1053     1615 2
: 1054     1616 2
: 1055     1617 2
: 1056     1618 2
: 1057     1619 2
: 1058     1620 2
: 1059     1621 2
: 1060     1622 2
: 1061     1623 2
: 1062     1624 2
: 1063     1625 2
: 1064     1626 2
: 1065     1627 2
: 1066     1628 2
: 1067     1629 2
: 1068     1630 2
: 1069     1631 2
: 1070     1632 2
: 1071     1633 2
: 1072     1634 2
: 1073     1635 2
: 1074     1636 2
: 1075     1637 2
: 1076     1638 2
: 1077     1639 2
: 1078     1640 2
: 1079     1641 2
: 1080     1642 2
: 1081     1643 2
: 1082     1644 2
: 1083     1645 2
: 1084     1646 2
: 1085     1647 2
: 1086     1648 2
: 1087     1649 2
: 1088     1650 2
: 1089     1651 2
: 1090     1652 2
: 1091     1653 2
: 1092     1654 2
: 1093     1655 2
: 1094     1656 2
: 1095     1657 2
: 1096     1658 2
: 1097     1659 2
: 1098     1660 2
: 1099     1661 2
: 1100     1662 2
: 1101     1663 2
: 1102     1664 2
: 1103     1665 2
: 1104     1666 2
: 1105     1667 2
: 1106     1668 2
: 1107     1669 2
: 1108     1670 2
: 1109     1671 2
: 1110     1672 2
: 1111     1673 2
: 1112     1674 2
: 1113     1675 2
: 1114     1676 2
: 1115     1677 2
: 1116     1678 2
: 1117     1679 2
: 1118     1680 2
: 1119     1681 2
: 1120     1682 2
: 1121     1683 2
: 1122     1684 2
: 1123     1685 2
: 1124     1686 2
: 1125     1687 2
: 1126     1688 2
: 1127     1689 2
: 1128     1690 2
: 1129     1691 2
: 1130     1692 2
: 1131     1693 2
: 1132     1694 2
: 1133     1695 2
: 1134     1696 2
: 1135     1697 2
: 1136     1698 2
: 1137     1699 2
: 1138     1700 2
: 1139     1701 2
: 1140     1702 2
: 1141     1703 2
: 1142     1704 2
: 1143     1705 2
: 1144     1706 2
: 1145     1707 2
: 1146     1708 2
: 1147     1709 2
: 1148     1710 2
: 1149     1711 2
: 1150     1712 2
: 1151     1713 2
: 1152     1714 2
: 1153     1715 2
: 1154     1716 2
: 1155     1717 2
: 1156     1718 2
: 1157     1719 2
: 1158     1720 2
: 1159     1721 2
: 1160     1722 2
: 1161     1723 2
: 1162     1724 2
: 1163     1725 2
: 1164     1726 2
: 1165     1727 2
: 1166     1728 2
: 1167     1729 2
: 1168     1730 2
: 1169     1731 2
: 1170     1732 2
: 1171     1733 2
: 1172     1734 2
: 1173     1735 2
: 1174     1736 2
: 1175     1737 2
: 1176     1738 2
: 1177     1739 2
: 1178     1740 2
: 1179     1741 2
: 1180     1742 2
: 1181     1743 2
: 1182     1744 2
: 1183     1745 2
: 1184     1746 2
: 1185     1747 2
: 1186     1748 2
: 1187     1749 2
: 1188     1750 2
: 1189     1751 2
: 1190     1752 2
: 1191     1753 2
: 1192     1754 2
: 1193     1755 2
: 1194     1756 2
: 1195     1757 2
: 1196     1758 2
: 1197     1759 2
: 1198     1760 2
: 1199     1761 2
: 1200     1762 2
: 1201     1763 2
: 1202     1764 2
: 1203     1765 2
: 1204     1766 2
: 1205     1767 2
: 1206     1768 2
: 1207     1769 2
: 1208     1770 2
: 1209     1771 2
: 1210     1772 2
: 1211     1773 2
: 1212     1774 2
: 1213     1775 2
: 1214     1776 2
: 1215     1777 2
: 1216     1778 2
: 1217     1779 2
: 1218     1780 2
: 1219     1781 2
: 1220     1782 2
: 1221     1783 2
: 1222     1784 2
: 1223     1785 2
: 1224     1786 2
: 1225     1787 2
: 1226     1788 2
: 1227     1789 2
: 1228     1790 2
: 1229     1791 2
: 1230     1792 2
: 1231     1793 2
: 1232     1794 2
: 1233     1795 2
: 1234     1796 2
: 1235     1797 2
: 1236     1798 2
: 1237     1799 2
: 1238     1800 2
: 1239     1801 2
: 1240     1802 2
: 1241     1803 2
: 1242     1804 2
: 1243     1805 2
: 1244     1806 2
: 1245     1807 2
: 1246     1808 2
: 1247     1809 2
: 1248     1810 2
: 1249     1811 2
: 1250     1812 2
: 1251     1813 2
: 1252     1814 2
: 1253     1815 2
: 1254     1816 2
: 1255     1817 2
: 1256     1818 2
: 1257     1819 2
: 1258     1820 2
: 1259     1821 2
: 1260     1822 2
: 1261     1823 2
: 1262     1824 2
: 1263     1825 2
: 1264     1826 2
: 1265     1827 2
: 1266     1828 2
: 1267     1829 2
: 1268     1830 2
: 1269     1831 2
: 1270     1832 2
: 1271     1833 2
: 1272     1834 2
: 1273     1835 2
: 1274     1836 2
: 1275     1837 2
: 1276     1838 2
: 1277     1839 2
: 1278     1840 2
: 1279     1841 2
: 1280     1842 2
: 1281     1843 2
: 1282     1844 2
: 1283     1845 2
: 1284     1846 2
: 1285     1847 2
: 1286     1848 2
: 1287     1849 2
: 1288     1850 2
: 1289     1851 2
: 1290     1852 2
: 1291     1853 2
: 1292     1854 2
: 1293     1855 2
: 1294     1856 2
: 1295     1857 2
: 1296     1858 2
: 1297     1859 2
: 1298     1860 2
: 1299     1861 2
: 1300     1862 2
: 1301     1863 2
: 1302     1864 2
: 1303     1865 2
: 1304     1866 2
: 1305     1867 2
: 1306     1868 2
: 1307     1869 2
: 1308     1870 2
: 1309     1871 2
: 1310     1872 2
: 1311     1873 2
: 1312     1874 2
: 1313     1875 2
: 1314     1876 2
: 1315     1877 2
: 1316     1878 2
: 1317     1879 2
: 1318     1880 2
: 1319     1881 2
: 1320     1882 2
: 1321     1883 2
: 1322     1884 2
: 1323     1885 2
: 1324     1886 2
: 1325     1887 2
: 1326     1888 2
: 1327     1889 2
: 1328     1890 2
: 1329     1891 2
: 1330     1892 2
: 1331     1893 2
: 1332     1894 2
: 1333     1895 2
: 1334     1896 2
: 1335     1897 2
: 1336     1898 2
: 1337     1899 2
: 1338     1900 2
: 1339     1901 2
: 1340     1902 2
: 1341     1903 2
: 1342     1904 2
: 1343     1905 2
: 1344     1906 2
: 1345     1907 2
: 1346     1908 2
: 1347     1909 2
: 1348     1910 2
: 1349     1911 2
: 1350     1912 2
: 1351     1913 2
: 1352     1914 2
: 1353     1915 2
: 1354     1916 2
: 1355     1917 2
: 1356     1918 2
: 1357     1919 2
: 1358     1920 2
: 1359     1921 2
: 1360     1922 2
: 1361     1923 2
: 1362     1924 2
: 1363     1925 2
: 1364     1926 2
: 1365     1927 2
: 1366     1928 2
: 1367     1929 2
: 1368     1930 2
: 1369     1931 2
: 1370     1932 2
: 1371     1933 2
: 1372     1934 2
: 1373     1935 2
: 1374     1936 2
: 1375     1937 2
: 1376     1938 2
: 1377     1939 2
: 1378     1940 2
: 1379     1941 2
: 1380     1942 2
: 1381     1943 2
: 1382     1944 2
: 1383     1945 2
: 1384     1946 2
: 1385     1947 2
: 1386     1948 2
: 1387     1949 2
: 1388     1950 2
: 1389     1951 2
: 1390     1952 2
: 1391     1953 2
: 1392     1954 2
: 1393     1955 2
: 1394     1956 2
: 1395     1957 2
: 1396     1958 2
: 1397     1959 2
: 1398     1960 2
: 1399     1961 2
: 1400     1962 2
: 1401     1963 2
: 1402     1964 2
: 1403     1965 2
: 1404     1966 2
: 1405     1967 2
: 1406     1968 2
: 1407     1969 2
: 1408     1970 2
: 1409     1971 2
: 1410     1972 2
: 1411     1973 2
: 1412     1974 2
: 1413     1975 2
: 1414     1976 2
: 1415     1977 2
: 1416     1978 2
: 1417     1979 2
: 1418     1980 2
: 1419     1981 2
: 1420     1982 2
: 1421     1983 2
: 1422     1984 2
: 1423     1985 2
: 1424     1986 2
: 1425     1987 2
: 1426     1988 2
: 1427     1989 2
: 1428     1990 2
: 1429     1991 2
: 1430     1992 2
: 1431     1993 2
: 1432     1994 2
: 1433     1995 2
: 1434     1996 2
: 1435     1997 2
: 1436     1998 2
: 1437     1999 2
: 1438     2000 2
: 1439     2001 2
: 1440     2002 2
: 1441     2003 2
: 1442     2004 2
: 1443     2005 2
: 1444     2006 2
: 1445     2007 2
: 1446     2008 2
: 1447     20
```



```

669      1230      2      Establish the input buffer address.
670      1231      2
671      1232      2
672      1233      2      Input_rab [rab$l_ubf] = emb ;          ! Load input buffer addr. in RAB
673      1234      2
674      1235      2
675      1236      2      OPEN AND CONNECT ---
676      1237      2      To the input error log file. Exit immediately if any errors
677      1238      2      are detected. The error handlers will have been invoked if an
678      1239      2      error occurred, and the user will have been notified. The error
679      1240      2      message used by the LOG_FILENAME routine is drawn from the
680      1241      2      CTX field of the FAB or INPUT_RAB as needed.
681      1242      2
682      1243      2
683      1244      2      Fab [fab$l_ctx] = msg$ openin;          ! Specify the error message
684      P 1245      2      CALL_FUNCTION ( $open (fab=fab,          ! OPEN the input file
685      1246      2      err=log_filename) );
686      P 1247      2      CALL_FUNCTION ( $connect (rab=input_rab, ! CONNECT to input file
687      1248      2      err=log_filename) );
688      1249      2
689      1250      2
690      1251      2
691      1252      2
692      1253      2
693      1254      2      INITIALIZE OUTPUT FILES --
694      1255      2      The processing of the output files has been deferred until now
695      1256      2      so that a fully parsed input file name would be available (as
696      1257      2      a related file name) for default file name components.
697      1258      2
698      1259      2
699      1260      2      If .input file count eq 0 then          ! If this is the first pass
700      1261      2      CALL_FUNCTION ( parse_output_files(Lstlun) ); ! then open the output files
701      1262      2
702      1263      2      Input_file_count = .input_file_count + 1;      ! Count the number of files we process.
703      1264      2
704      1265      2      Lstlun = .syecom[sye$l_lstlun];          ! Initialize the fortran logical unit number
705      1266      2
706      1267      2
707      1268      2
708      1269      2      RESET THE RECORD COUNTERS--
709      1270      2      Reset the file-relative record number such that records in each
710      1271      2      file will be numbered in ascending order beginning with one.
711      1272      2      Reset the selected count so it is also on a per-file basis.
712      1273      2
713      1274      2
714      1275      2      Syecom[sye$l_reccnt] = 0;          ! Reset the record number
715      1276      2      Selected = 0;          ! Reset file select count
716      1277      2

```

```
718 1278 2 |
719 1279 2 | READ AND LOOP UNTIL EOF OR ERROR ---
720 1280 2 | While status is true, get a record from the input file.
721 1281 2 |   Increment the SYECOM[SYE$L_RECCNT] which is the record count.
722 1282 2 |   If the user command specified this type of record needs processing then
723 1283 2 |     Increment record selected count.
724 1284 2 |     If /BINARY specified, then
725 1285 2 |       write the record to the specified binary file,
726 1286 2 |       else go process the record(error packet).
727 1287 2 |       else write the error packet to the rejected records file if specified.
728 1288 2 |   End while
729 1289 2 |
730 1290 2 |
731 1291 2 | While status = $GET (rab=input_rab, err=log_filename) do ! Read a record
732 1292 3 | BEGIN
733 1293 3 |   Syecom[sye$l_reccnt] = .syecom[sye$l_reccnt] + 1; ! Update the record number
734 1294 3 |
735 1295 3 |   If record_selected () then | Process this record?
736 1296 4 |     BEGIN | Yes
737 1297 4 |       Selected = .selected + 1; | Count how many we process
738 1298 4 |       If .option_flag[opt$V_binary_qual] then | If /BINARY write packet
739 1299 5 |         CALL_FUNCTION ( write_binary (emb, output_rab) )
740 1300 4 |       Else
741 1301 4 |         CALL_FUNCTION ( process_packet () ); ! Analyze packet
742 1302 4 |       End
743 1303 3 |     Else
744 1304 3 |       If .option_flag[opt$V_rejected_qual] | ! If /REJECTED write packet
745 1305 3 |       then CALL_FUNCTION ( write_binary (emb, rejected_rab) );
746 1306 3 |
747 1307 3 |
748 1308 3 | Determine if the end value for a '/entry=end' was found.
749 1309 3 | Yes, set up status and exit as if end of file.
750 1310 3 |
751 1311 3 | If .syecom[sye$b_end_value] then
752 1312 4 |   Begin
753 1313 4 |     Status = RMS$_EOF ;
754 1314 4 |     Exitloop ;
755 1315 3 |   End ;
756 1316 2 | END;
757 1317 2 |
758 1318 2 | Total_selected = .total_selected + .selected; ! Accumulate totals
759 1319 2 | Total_rejected = .total_rejected + (.syecom[sye$l_reccnt] - .selected);
760 1320 2 |
761 1321 2 |
762 1322 2 |
763 1323 2 | CHECK STATUS AT END OF LOOP ---
764 1324 2 | Now check the return status to make sure it was a normal EOF. If not,
765 1325 2 | notify the user.
766 1326 2 |
767 1327 2 |
768 1328 3 | If not (.status eql rms$_eof) | If any status other than
769 1329 2 | then return .status; | expected eof, return it
770 1330 2 |
771 1331 2 |
772 1332 2 | Indicate that end of file occurred.
773 1333 2 |
774 1334 2 | Syecom[sye$b_eof_flag] = true ;
```



```
776 1335 2 |
777 1336 2 | Display MSCP messages
778 1337 2 | Determine if a full report should be generated, so that the MSCP
779 1338 2 | messages can be de-queued and output. (This must be done after EOF
780 1339 2 | because the number of entries for a given cmd ref number is unknown
781 1340 2 | due to the number of retries for the I/O, etc.).
782 1341 2 | *****
783 1342 2 | Brief reports are handled in the root and don't seem to require this
784 1343 2 | call back to de-queue any information.??????
785 1344 2 | Summary information is updated when the first part of an MSCP message
786 1345 2 | is seen. Summaries for device rollup worked before this code was put in.
787 1346 2 |
788 1347 2 | If .parser_data[erl$b_rpt_type] EQLU FULL_REP
789 1348 2 | Then
790 1349 2 |
791 1350 2 |     Set up the class/type and attempt to load the ERFPROC1 image. If the
792 1351 2 |     image was already loaded the xfer address will be returned.
793 1352 2 |
794 1353 2 |     Begin
795 1354 2 |     Class = 0 ;
796 1355 2 |     Type = EMB$C_SP ;
797 1356 2 |
798 1357 2 |     Worst_error = Image_loader ( type, class, xfer_addr );
799 1358 2 |
800 1359 2 |     If .Xfer_addr NEQU 0
801 1360 2 |     Then
802 1361 2 |
803 1362 2 |         Call the device dependent module to produce a full report.
804 1363 2 |         (Record_size is passed as a count of 1 and record number is unknown
805 1364 2 |         at this time and should not matter, because when ERLLOGSTS sees that
806 1365 2 |         EOF was seen it will call the DUDRIVER_MSCP_DQ routine to output the
807 1366 2 |         remainder of the MSCP message information).
808 1367 2 |
809 1368 2 |     BEGIN
810 1369 2 |     Syecom[syel_options] = %c'S';
811 1370 2 |     Exec_image ( Xfer_addr, Lstlun, %REF(1), syecom[syel_reccnt],
812 1371 2 |                 AD('S') );
813 1372 2 |     End;
814 1373 2 | End ;
815 1374 2 |
816 1375 2 |
817 1376 2 | Determine whether any of the specified enrties were found.
818 1377 2 | If no entries found. Output an informational message for the user.
819 1378 2 |
820 1379 2 |
821 1380 2 | If .total_selected EQL 0 then signal (erf_notfound) ;
822 1381 2 |
823 1382 2 |
824 1383 2 | CLOSE ---
825 1384 2 |     Input file processing is now complete. Revise the stored error
826 1385 2 |     message (which is passed to the error routine via the 'user context'
827 1386 2 |     field [CTX] of the FAB) and $CLOSE the input file.
828 1387 2 |
829 1388 2 |
830 1389 2 | Fab [fab$l_ctx] = msg$_closein;          ! Assign error message
831 1390 2 | CALL_FUNCTION ( $close (fab=.fab, err=log_filename) ); ! Close the input file
832 1391 2 |
```

```

: 833      1392 2
: 834      1393 2 If .option_flag[opt$v_log_qual] then          ! If /LOG requested
: 835      1394 2 BEGIN
: 836      1395 2 Desc [0] = .input_nam [nam$b_rsl];           ! then notify the user
: 837      1396 2 Desc [1] = .input_nam [nam$l_rsl];           ! with file name and counts
: 838      1397 2 Signal (erf_input, 3, desc, .selected, .syecom[sye$l_recnt] - .selected);
: 839      1398 2 END;
: 840      1399 2
: 841      1400 2
: 842      1401 2
: 843      1402 2 Return true;
: 844      1403 2
: 845      1404 1 END;
```

```

                                .PSECT $PLIT,NOWRT,NOEXE, PIC,2
00 00 00 53 00044 P.AAH: .ASCII \S\<0><0><0>
00000001 00048 P.AAG: .LONG 1
00000000' 0004C .ADDRESS P.AAH
```

```

                                .PSECT $OWNS,NOEXE, PIC,2
00000001 00004 CLASS: .BLKB 2
00006 .BLKB 2
00008 FIRST_TIME:
0000C TYPE: .LONG 1
0000E .BLKB 2
00010 XFER_ADDR: .BLKB 4
```

```

.EXTRN SYS$OPEN, SYS$CONNECT
.EXTRN SYS$GET
```

```

.PSECT $CODE,NOWRT, PIC,2
```

```

OFFC 00000 PROCESS_FILE:
5B 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 : 1186
5A 00000000G 00 9E 00009 MOVAB EMB, R11
59 00000000G 00 9E 00010 MOVAB LIB$SIGNAL, R10
58 00000000G 00 9E 00017 MOVAB OPTION_FLAG, R9
57 00000000G 00 9E 0001E MOVAB INPUT_RAB, R8
56 00000000G 00 9E 00025 MOVAB LOG_FILENAME, R7
55 00000000' 00 9E 0002C MOVAB SYE$COM, R6
54 00000000' 00 9E 00033 MOVAB SELECTED, R5
5E 00000000' 0C C2 0003A MOVAB LSTLUN, R4
24 A8 00000000 6B 9E 0003D SUBL2 #12, SP
53 00000000 04 AC D0 00041 MOVAB EMB, INPUT_RAB+36 : 1233
18 A3 0008109A 8F D0 00045 MOVL FAB, R3 : 1244
00000000G 00 0088 8F BB 0004D MOVL #528538, 24(R3)
77 00 02 FB 00051 PUSHF #*M<R3,R7> : 1246
50 E9 00058 CALLS #2, SYS$OPEN
57 DD 0005B BLBC STATUS, 6$
58 DD 0005D PUSHF R7 : 1248
PUSHF R8
```


00000000G	00		02	FB	0005F	CALLS	#2, SYSSCONNECT	:		
	69		50	E9	00066	BLBC	STATUS, 6\$:		
		AC	A5	D5	00069	TSTL	INPUT_FILE_COUNT	:	1260	
			0C	12	0006C	BNEQ	1\$:		
			54	DD	0006E	PUSHL	R4	:	1261	
00000000G	00		01	FB	00070	CALLS	#1, PARSE_OUTPUT_FILES	:		
	58		50	E9	00077	BLBC	STATUS, 6\$:		
		AC	A5	D6	0007A	1\$: INCL	INPUT_FILE_COUNT	:	1263	
	64	27	A6	D0	0007D	MOVL	SYECOM+39, -LSTLUN	:	1265	
			66	D4	00081	CLRL	SYECOM	:	1275	
			65	D4	00083	CLRL	SELECTED	:	1276	
			57	DD	00085	2\$: PUSHL	R7	:	1291	
			58	DD	00087	PUSHL	R8	:		
00000000G	00		02	FB	00089	CALLS	#2, SYSSGET	:		
	52		50	D0	00090	MOVL	R0, STATUS	:		
	4B		52	E9	00093	BLBC	STATUS, 8\$:		
			66	D6	00096	INCL	SYECOM	:	1293	
00000000G	00		00	FB	00098	CALLS	#0, RECORD_SELECTED	:	1295	
	1A		50	E9	0009F	BLBC	R0, 4\$:		
			65	D6	000A2	INCL	SELECTED	:	1297	
	50		69	D0	000A4	MOVL	OPTION_FLAG, R0	:	1298	
08	60		01	E1	000A7	BBC	#1, (R0), 3\$:		
		00000000G	00	9F	000AB	PUSHAB	OUTPUT_RAB	:	1299	
			16	11	000B1	BRB	5\$:		
00000000V	00		00	FB	000B3	3\$: CALLS	#0, PROCESS_PACKET	:	1301	
			16	11	000BA	BRB	6\$:		
	50		69	D0	000BC	4\$: MOVL	OPTION_FLAG, R0	:	1304	
13	60		0A	E1	000BF	BBC	#10, (R0), 7\$:		
		00000000G	00	9F	000C3	PUSHAB	REJECTED_RAB	:	1305	
			5B	DD	000C9	5\$: PUSHL	R11	:		
00000000V	00		02	FB	000CB	CALLS	#2, WRITE_BINARY	:		
	01		50	E8	000D2	6\$: BLBS	STATUS, 7\$:		
				04	000D5	RET		:		
	AB	1E	A6	E9	000D6	7\$: BLBC	SYECOM+30, 2\$:	1311	
	52	0001827A	8F	D0	000DA	MOVL	#98938, STATUS	:	1313	
	50		65	D0	000E1	8\$: MOVL	SELECTED, R0	:	1318	
	24		50	C0	000E4	ADDL2	R0, TOTAL_SELECTED	:		
50	66		50	C3	000E8	SUBL3	R0, SYECOM, R0	:	1319	
	28		50	C0	000EC	ADDL2	R0, TOTAL_REJECTED	:		
0001827A	8F		52	D1	000F0	CMPL	STATUS, #98938	:	1328	
			04	13	000F7	BEQL	9\$:		
	50		52	D0	000F9	MOVL	STATUS, R0	:	1329	
				04	000FC	RET		:		
	1D		01	90	000FD	9\$: MOVB	#1, SYECOM+29	:	1334	
	50	00000000G	00	D0	00101	MOVL	PARSER_DATA, R0	:	1347	
	02		60	91	00108	CMPB	(R0), #2	:		
			44	12	0010B	BNEQ	10\$:		
		04	A4	B4	0010D	CLRW	CLASS	:	1354	
	0C	A4	8F	9B	00110	MOVZBW	#99, TYPE	:	1355	
		10	A4	9F	00115	PUSHAB	XFER_ADDR	:	1357	
		04	A4	9F	00118	PUSHAB	CLASS	:		
		0C	A4	9F	0011B	PUSHAB	TYPE	:		
00000000G	00		03	FB	0011E	CALLS	#3, IMAGE_LOADER	:		
00000000G	00		50	D0	00125	MOVL	R0, WORST_ERROR	:		
		10	A4	D5	0012C	TSTL	XFER_ADDR	:	1359	
			20	13	0012F	BEQL	10\$:		
	2B	A6	53	8F	9A	00131	MOVZBL	#83, SYECOM+43	:	1369

		00000000'	00	9F	00136	PUSHAB	P.AAG	:	1371
			56	DD	0013C	PUSHL	R6	:	1370
08	AE		01	DD	0013E	MOVL	#1, 8(SP)	:	
		08	AE	9F	00142	PUSHAB	8(SP)	:	
			54	DD	00145	PUSHL	R4	:	
		10	A4	9F	00147	PUSHAB	XFER_ADDR	:	
00000000G	00		05	FB	0014A	CALLS	#5, EXEC_IMAGE	:	
		24	A5	D5	00151	TSTL	TOTAL_SELECTED	:	1380
			09	12	00154	BNEQ	11\$:	
		00000000G	8F	DD	00156	PUSHL	#ERF_NOTFOUND	:	
	6A		01	FB	0015C	CALLS	#1, CIB\$SIGNAL	:	
18	A3	00081052	8F	DD	0015F	MOVL	#528466, 24(R3)	:	1389
		0088	8F	BB	00167	PUSHR	#^M<R3,R7>	:	1390
00000000G	00		02	FB	0016B	CALLS	#2, SY\$CLOSE	:	
	31		50	E9	00172	BLBC	STATUS, 13\$:	
	50		69	DD	00175	MOVL	OPTION_FLAG, R0	:	1393
			60	95	00178	TSTB	(R0)	:	
			27	18	0017A	BGEQ	12\$:	
04	AE	00000000G	00	9A	0017C	MOVZBL	INPUT_NAM+3, DESC	:	1395
08	AE	00000000G	00	DD	00184	MOVL	INPUT_NAM+4, DESC+4	:	1396
	50		65	DD	0018C	MOVL	SELECTED, R0	:	1397
7E	66		50	C3	0018F	SUBL3	R0, SYECOM, -(SP)	:	
			50	DD	00193	PUSHL	R0	:	
		0C	AE	9F	00195	PUSHAB	DESC	:	
			03	DD	00198	PUSHL	#3	:	
		00000000G	8F	DD	0019A	PUSHL	#ERF_INPUT	:	
	6A		05	FB	001A0	CALLS	#5, CIB\$SIGNAL	:	
	50		01	DD	001A3	MOVL	#1, R0	:	1402
			04	001A6	13\$:	RET		:	1404

; Routine Size: 423 bytes, Routine Base: \$CODE + 0235


```

: 847      1405 1 Routine PROCESS_PACKET =
: 848      1406 2 BEGIN
: 849      1407 2 ++
: 850      1408 2 Functional description
: 851      1409 2
: 852      1410 2 This routine determines which dispatcher to call for processing
: 853      1411 2 the specified report type. Any errors encountered will be passed
: 854      1412 2 back to the caller.
: 855      1413 2
: 856      1414 2 Calling sequence
: 857      1415 2
: 858      1416 2 Process_packet ( )
: 859      1417 2
: 860      1418 2 Input parameters
: 861      1419 2
: 862      1420 2
: 863      1421 2 Output parameters
: 864      1422 2
: 865      1423 2 None
: 866      1424 2
: 867      1425 2 Routine value
: 868      1426 2
: 869      1427 2 Worst error is returned.
: 870      1428 2
: 871      1429 2 ----
: 872      1430 2
: 873      1431 2 Literal
: 874      1432 2 No_full = 0 ;
: 875      1433 2
: 876      1434 2 Local
: 877      1435 2 Status;
: 878      1436 2
: 879      1437 2 Global
: 880      1438 2 Brief_xfer_addr;
: 881      1439 2
: 882      1440 2 syecom[sye$l_record_size] = .input_rab [rab$w_rsz];
: 883      1441 2
: 884      1442 2 If (NOT .unknown_entry) then ! If not an unknown entry then
: 885      1443 2 Begin
: 886      1444 2 Case .parser_data[erl$b_rpt_type] ! Case on report type value
: 887      1445 2 From 0 to REG_DUMP_REP of
: 888      1446 2 Set
: 889      1447 2
: 890      1448 2 [No_full]:
: 891      1449 2 If .option_flag[opt$v_summary_qual] then
: 892      1450 2 CALL_FUNCTION ( full_dispatcher ( ) );
: 893      1451 2
: 894      1452 2
: 895      1453 2 [Brief_rep]: ! Go output a Brief report
: 896      1454 2 Begin
: 897      1455 2 If .Brief_xfer_addr EQL 0 then
: 898      1456 2 Begin
: 899      1457 2 Status = map_image (AD ('SYS$SYSTEM:ERFBRIEF.EXE'),brief_xfer_addr);
: 900      1458 2 If NOT .status then return true;
: 901      1459 2 End;
: 902      1460 2
: 903      1461 2 Syecom[sye$l_options] = %c'B';
```

```
.. 904      1462  4      Exec_image( Brief_xfer_addr,
905      1463  4          Lstlun,
906      1464  4          AD('B'),
907      1465  4          syecom[sye$l_record_size],
908      1466  4          %REF(.option_flag[opt$v_summary_qual]),
909      1467  4          .Summary_flag) ;
910      1468  3      End;
911      1469  3
912      1470  3      [Full_rep]:          ! Go output a Full report
913      1471  4      Begin
914      1472  4      |
915      1473  4      | Determine whether the fortran text commons (giocommon, opcodes, modes)
916      1474  4      | have been initialized, if not then call the init_commons routine to
917      1475  4      | initialize them. Then call the full dispatcher.
918      1476  4      |
919      1477  4      | If NOT .inited_commons then CALL FUNCTION (init_commons()) ;
920      1478  4      | CALL_FUNCTION ( Full_dispatcher () );
921      1479  3      End ;
922      1480  3
923      1481  3      [Reg_dump_rep]:          ! Go output a Register Dump report
924      1482  3      Begin
925      1483  4      |
926      1484  4      | If .Brief_xfer_addr EQL 0 then
927      1485  5      | Begin
928      1486  5      | Status = map_image (AD ('SYS$SYSTEM:ERFBRIEF.EXE'),brief_xfer_addr);
929      1487  5      | If NOT .status then return true;
930      1488  4      | End;
931      1489  4      |
932      1490  4      | Syecom[sye$l_options] = %c'C';
933      1491  4      | Exec_image( Brief_xfer_addr,
934      1492  4      |          Lstlun,
935      1493  4      |          AD('C'),
936      1494  4      |          syecom[sye$l_record_size],
937      1495  4      |          %REF(.option_flag[opt$v_summary_qual]),
938      1496  4      |          .Summary_flag) ;
939      1497  4      |
940      1498  3      End;
941      1499  3
942      1500  3      [OUTRANGE]:
943      1501  3      | Signal (erf_invreptyp);
944      1502  3      |
945      1503  3      TES;
946      1504  3      End
947      1505  2      Else
948      1506  2      | If .parser_data[erl$b_rpt_type] NEQU NO_FULL then
949      1507  2      | Unknown_dispatcher ();          ! Call unknown dispatcher
950      1508  2      |
951      1509  2      |
952      1510  2      |
953      1511  2      | If /summary was specified then call summary_dispatcher.
954      1512  2      |
955      1513  2      | If .option_flag [opt$v_summary_qual] then
956      1514  3      | Begin
957      1515  3      | If (.summary_flag[sum$v_device] OR .summary_flag[sum$v_all_summ]) then
958      1516  3      | Exec_image ( Summary_dispatcher_addr, Lstlun, %REF(dev_summ_upd)) ;
959      1517  3      |
960      1518  3      | If (.summary_flag[sum$v_histogram] OR .summary_flag[sum$v_all_summ]) then
```



```
: 961      1519 3 Exec_image ( Summary_dispatcher_addr, Lstlun, %REF(histo_summ_upd)) ;
: 962      1520 2 End;
: 963      1521 2
: 964      1522 2 Return true;
: 965      1523 1 End;
```

```
42 46 52 45 3A 4D 45 54 53 59 53 24 53 59 53 00050 P.AAJ: .PSECT $SPLIT,NOWRT,NOEXE, PIC,2
      00 45 58 45 2E 46 45 49 52 0005F .ASCII \SYSS$SYSTEM:ERFBRIEF.EXE\<0>
      00000017 00068 P.AAI: .LONG 23
      00000000' 0006C .ADDRESS P.AAJ
      00 00 00 42 00070 P.AAL: .ASCII \B\<0><0><0>
      00000001 00074 P.AAK: .LONG 1
      00000000' 00078 .ADDRESS P.AAL
42 46 52 45 3A 4D 45 54 53 59 53 24 53 59 53 0007C P.AAN: .ASCII \SYSS$SYSTEM:ERFBRIEF.EXE\<0>
      00 45 58 45 2E 46 45 49 52 0008B .LONG 23
      00000017 00094 P.AAM: .ADDRESS P.AAN
      00000000' 00098 .ASCII \C\<0><0><0>
      00 00 00 43 0009C P.AAP: .LONG 1
      00000001 000A0 P.AAO: .ADDRESS P.AAP
      00000000' 000A4
```

.PSECT \$GLOBAL\$,NOEXE, PIC,2

000B8 BRIEF_XFER_ADDR::
.BCKB 4

.PSECT \$CODE,NOWRT, PIC,2

03FC 00000 PROCESS_PACKET:

```
59 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9 : 1405
58 00000000G 00 9E 00009 MOVAB MAP_IMAGE, R9
57 00000000' 00 9E 00010 MOVAB EXEC_IMAGE, R8
56 00000000G 00 9E 00017 MOVAB LSTLON, R7
55 00000000' 00 9E 0001E MOVAB SUMMARY_FLAG, R6
54 00000000G 00 9E 00025 MOVAB P.AAI, R5
53 00000000G 00 9E 0002C MOVAB OPTION_FLAG, R4
52 00000000' 00 9E 00033 MOVAB SYECOM+35, R3
5E 00000000' 04 C2 0003A MOVAB BRIEF_XFER_ADDR, R2
63 00000000G 00 3C 0003D SUBL2 #4, SP
50 00000000G 00 D0 00044 MOVZWL INPUT_RAB+34, SYECOM+35 : 1440
03 00000000G 00 00 0004B MOVL PARSE_DATA, R0 : 1444
      F8 A2 E9 0004B BLBC UNKNOWN_ENTRY, 1$ : 1442
      009E 31 0004F BRW 13$
0067 003 00 60 8F 00052 1$: CASEB (R0), #0, #3 : 1444
      004C 0018 00056 2$: .WORD 4$-2$,-
      0021 00056 3$: 5$-2$,-
      00000000G 00 00000000G 8F DD 0005E PUSHL #ERF_INVREPTYP : 1501
      00000000G 00 00000000G 01 FB 00064 CALLS #1, CIB$SIGNAL
      008D 31 0006B 3$: BRW 14$
```

					64	D0	0006E	4\$:	MOVL	OPTION_FLAG, R0	1449
					0E	E1	00071		BBC	#14, (R0), 3\$	
					3B	11	00075		BRB	8\$	1450
					62	D5	00077	5\$:	TSTL	BRIEF_XFER_ADDR	1455
					0D	12	00079		BNEQ	6\$	
					52	DD	0007B		PUSHL	R2	1457
					55	DD	0007D		PUSHL	R5	
					02	FB	0007F		CALLS	#2, MAP_IMAGE	
					50	D0	00082		MOVL	R0, STATUS	
					51	E9	00085		BLBC	STATUS, 10\$	1458
					8F	9A	00088	6\$:	MOVZBL	#66, SYECOM+43	1461
					66	DD	0008D		PUSHL	SUMMARY_FLAG	1467
					64	D0	0008F		MOVL	OPTION_FLAG, R0	1466
					0E	EF	00092		EXTZV	#14, #T, (R0), 4(SP)	
					AE	9F	00098		PUSHAB	4(SP)	
					53	DD	0009B		PUSHL	R3	1465
					A5	9F	0009D		PUSHAB	P.AAK	1464
					45	11	000A0		BRB	12\$	1462
					C2	E8	000A2	7\$:	BLBS	INITED_COMMONS, 8\$	1477
					00	FB	000A7		CALLS	#0, INIT_COMMONS	
					50	E8	000AE		BLBS	STATUS, 8\$	
					04	00	000B1		RET		
					00	FB	000B2	8\$:	CALLS	#0, FULL_DISPATCHER	1478
					50	E8	000B9		BLBS	STATUS, T4\$	
					04	00	000BC		RET		
					62	D5	000BD	9\$:	TSTL	BRIEF_XFER_ADDR	1484
					0E	12	000BF		BNEQ	11\$	
					52	DD	000C1		PUSHL	R2	1486
					A5	9F	000C3		PUSHAB	P.AAM	
					02	FB	000C6		CALLS	#2, MAP_IMAGE	
					50	D0	000C9		MOVL	R0, STATUS	
					51	E9	000CC	10\$:	BLBC	STATUS, 18\$	1487
					8F	9A	000CF	11\$:	MOVZBL	#67, SYECOM+43	1490
					66	DD	000D4		PUSHL	SUMMARY_FLAG	1496
					64	D0	000D6		MOVL	OPTION_FLAG, R0	1495
					0E	EF	000D9		EXTZV	#14, #T, (R0), 4(SP)	
					AE	9F	000DF		PUSHAB	4(SP)	
					53	DD	000E2		PUSHL	R3	1494
					A5	9F	000E4		PUSHAB	P.AAO	1493
					8F	BB	000E7	12\$:	PUSHR	#*M<R2,R7>	1491
					06	FB	000EB		CALLS	#6, EXEC_IMAGE	
					0B	11	000EE		BRB	14\$	1442
					60	95	000F0	13\$:	TSTB	(R0)	1506
					07	13	000F2		BEQL	14\$	
					00	FB	000F4		CALLS	#0, UNKNOWN_DISPATCHER	1507
					64	D0	000FB	14\$:	MOVL	OPTION_FLAG, R0	1513
					0E	E1	000FE		BBC	#14, (R0), 18\$	
					66	D0	00102		MOVL	SUMMARY_FLAG, R0	1515
					01	E0	00105		BBS	#1, (R0), 15\$	
					60	E9	00109		BLBC	(R0), 16\$	
					03	D0	0010C	15\$:	MOVL	#3, (SP)	1516
					8F	BB	0010F		PUSHR	#*M<R7,SP>	
					A2	9F	00113		PUSHAB	SUMMARY_DISPATCHER_ADDR	
					03	FB	00116		CALLS	#3, EXEC_IMAGE	
					66	D0	00119	16\$:	MOVL	SUMMARY_FLAG, R0	1518
					05	E0	0011C		BBS	#5, (R0), 17\$	
					60	E9	00120		BLBC	(R0), 18\$	

ERF
V04-000

Errorlog Report Formatter

I 4
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 31
(11)

6E		08	D0	00123	17\$:	MOVL	#8, (SP)	:	1519
	4080	8F	BB	00126		PUSHR	#^M<R7, SP>	:	
	D0	A2	9F	0012A		PUSHAB	SUMMARY_DISPATCHER_ADDR	:	
68		03	FB	0012D		CALLS	#3, EXEC_IMAGE	:	
50		01	D0	00130	18\$:	MOVL	#1, R0	:	1522
		04	00133			RET		:	1523

; Routine Size: 308 bytes, Routine Base: \$CODE + 03DC

```

: 967 1524 1 Routine FULL_DISPATCHER = ! Full report dispatcher
: 968 1525 2 BEGIN
: 969 1526 3 ++
: 970 1527 3 Functional description
: 971 1528 3
: 972 1529 3 This routine checks to see if the loadable module needed
: 973 1530 3 to process the error packet is available. If it is not then
: 974 1531 3 it tries to load the module. If successful then it call the module.
: 975 1532 3 Any errors encountered will be passed back to this routine.
: 976 1533 3
: 977 1534 3 Calling sequence
: 978 1535 3
: 979 1536 3 Full_dispatcher ()
: 980 1537 3
: 981 1538 3 Input parameters
: 982 1539 3
: 983 1540 3
: 984 1541 3 Output parameters
: 985 1542 3
: 986 1543 3 None
: 987 1544 3
: 988 1545 3 Routine value
: 989 1546 3
: 990 1547 3 Worst error is returned.
: 991 1548 3
: 992 1549 3 ----
: 993 1550 3
: 994 1551 3 Own
: 995 1552 3 Xfer_addr: LONG, ! Transfer address of the required image
: 996 1553 3 Class: WORD, ! Class of the image to be loaded
: 997 1554 3 Type: WORD; ! Type of the image to be loaded
: 998 1555 3
: 999 1556 3
: 1000 1557 3
: 1001 1558 3 If the packet entry type is a device error, device timeout, or
: 1002 1559 3 device attention then use the class and type specified in the packet as
: 1003 1560 3 class and type of the image to be loaded. Else use class = 0 and
: 1004 1561 3 type = entry type value.
: 1005 1562 3
: 1006 1563 3
: 1007 1564 3 If ( .emb[emb$w_hd_entry] EQLU EMB$C_DE ) OR
: 1008 1565 3 ( .emb[emb$w_hd_entry] EQLU EMB$C_DT ) OR
: 1009 1566 3 ( .emb[emb$w_hd_entry] EQLU EMB$C_DA )
: 1010 1567 3 Then
: 1011 1568 3 Begin
: 1012 1569 3 Type = .emb[emb$b_dv_type];
: 1013 1570 3 Class = .emb[emb$b_dv_class];
: 1014 1571 3 End
: 1015 1572 3 Else
: 1016 1573 3 Begin
: 1017 1574 3 Type = .emb[emb$w_hd_entry];
: 1018 1575 3 Class = 0;
: 1019 1576 3 End;
: 1020 1577 3
: 1021 1578 3
: 1022 1579 3 Try and load the image. If no image report error and return.
: 1023 1580 2
```



```
: 1024      1581 2 Worst_error = Image_loader ( type, class, xfer_addr );
: 1025      1582 2 If .Xfer_addr EQLO 0 then return .worst_error;
: 1026      1583 2
: 1027      1584 2
: 1028      1585 2 | The error packet entry type will determine which call to EXEC_IMAGE should
: 1029      1586 2 | be used in order to pass the necessary parameters to the loaded image
: 1030      1587 2 | for translation of the error packet.
: 1031      1588 2
: 1032      1589 2 Case .emb[emb$w_hd_entry] from 1 to EMB$C_UBC of
: 1033      1590 2 SET
: 1034      1591 2
: 1035      1592 2 [ EMB$C_DE,      | Device Error 1
: 1036      1593 2   EMB$C_BE,      | Bus Error 4
: 1037      1594 2   EMB$C_AW,      | Asynchronous Write Error 7
: 1038      1595 2   EMB$C_CS,      | Cold start (ie: SYSTEM BOOT) 32 %x20
: 1039      1596 2   34,      | NOT IN DEFINITION FILE 34 %x22
: 1040      1597 2   EMB$C_NF,      | New errlod.sys file created 35 %x23
: 1041      1598 2   EMB$C_WS,      | Warm start (ie: SYSTEM POWER RECOVERY) 36 %x24
: 1042      1599 2   EMB$C_CR,      | Fatal bugcheck 37 %x25
: 1043      1600 2   EMB$C_TS,      | Time stamp entry 38 %x26
: 1044      1601 2   EMB$C_SS,      | System service message 39 %x27
: 1045      1602 2   EMB$C_SBC,      | System bugcheck 40 %x28
: 1046      1603 2   EMB$C_OM,      | Operator message 41 %x29
: 1047      1604 2   EMB$C_NM,      | Network message 42 %x2A
: 1048      1605 2   EMB$C_DT,      | Device Timeout 96 %x60
: 1049      1606 2   EMB$C_UI,      | Undefined interrupt 97 %x61
: 1050      1607 2   EMB$C_DA,      | Asynchronous Device Attention 98 %x62
: 1051      1608 2   EMB$C_UBC ] :      | User bugcheck 112 %x70
: 1052      1609 2
: 1053      1610 2
: 1054      1611 2 | Determine if a full report should be generated.
: 1055      1612 2 | Call the device dependent module to produce a full report.
: 1056      1613 2 | Else return.
: 1057      1614 2
: 1058      1615 2 Begin
: 1059      1616 2 If .parser_data[erl$b_rpt_type] NEQ 0 then Exec_image ( Xfer_addr, Lstlun );
: 1060      1617 2 Return true;
: 1061      1618 2 End;
: 1062      1619 2
: 1063      1620 2 [ EMB$C_SP,      | Software Parameters 99 %x63
: 1064      1621 2   EMB$C_LM,      | Logged Message 100 %x64
: 1065      1622 2   EMB$C_LOGMSCP ] :      | MSCP message without UCB 101 %x65
: 1066      1623 2
: 1067      1624 2 Begin
: 1068      1625 2
: 1069      1626 2 | Determine if summary information should be updated.
: 1070      1627 2
: 1071      1628 2 If (.option_flag[opt$v_summary_qual]) AND
: 1072      1629 2   (.emb[emb$w_hd_entry] NEQO EMB$C_LOGMSCP)
: 1073      1630 2 Then
: 1074      1631 2 | Yes, call the device dependent module for summary updates.
: 1075      1632 2
: 1076      1633 2 BEGIN
: 1077      1634 2   Syecom[sys$l_options] = %c'R';
: 1078      1635 2   Exec_image ( Xfer_addr, Lstlun, syecom[sys$l_record_size],
: 1079      1636 2     syecom[sys$l_recnt], AD('R') );
: 1080      1637 2 END;
```

```
1081 1638 3
1082 1639 3
1083 1640 3
1084 1641 3
1085 1642 3
1086 1643 3
1087 1644 3
1088 1645 4
1089 1646 4
1090 1647 4
1091 1648 4
1092 1649 3
1093 1650 3
1094 1651 3
1095 1652 3
1096 1653 3
1097 1654 3
1098 1655 3
1099 1656 3
1100 1657 3
1101 1658 3
1102 1659 3
1103 1660 3
1104 1661 3
1105 1662 3
1106 1663 3
1107 1664 3
1108 1665 3
1109 1666 3
1110 1667 3
1111 1668 4
1112 1669 3
1113 1670 3
1114 1671 3
1115 1672 3
1116 1673 4
1117 1674 4
1118 1675 4
1119 1676 4
1120 1677 3
1121 1678 3
1122 1679 3
1123 1680 3
1124 1681 3
1125 1682 3
1126 1683 3
1127 1684 3
1128 1685 3
1129 1686 3
1130 1687 3
1131 1688 3
1132 1689 3
1133 1690 3
1134 1691 3
1135 1692 3
1136 1693 3
1137 1694 3

! If report type is not equal to NOFULL then call the device dependent
! module to produce a full report.
if .parser_data[erl$b_rpt_type] NEQ 0
then
  BEGIN
    Syecom[sys$l_options] = %c'S';
    Exec_image (Xfer_addr, Lstlun, syecom[sys$l_record_size],
                syecom[sys$l_reccnt], AD('S') );
  End;
  Return true;
End;

[ EMBSC_MC,           ! Machine check 2
  EMBSC_SA,           ! SBI Alert 5
  EMBSC_SE,           ! Soft ECC Error 6
  EMBSC_HE,           ! Hard ECC Error 8
  EMBSC_UBA,          ! 11/780 Unibus Adapter error 9
  EMBSC_UE,           ! 11/730 Unibus Error 11 %XB
  EMBSC_MBA,          ! 11/780 Massbus Adapter Error 12 %XC
  EMBSC_VM,           ! Volume mount 64 %X40
  EMBSC_VD ] :       ! Volume dismount 65 %X41

  Begin
    ! Determine if summary information should be updated.
    ! If (.parser_data[erl$b_rpt_type] EQL 0) AND
    !   (.option_flag[opt$v_summary_qual])
    Then
      ! Yes, call the device dependent module for summary updates
      ! and return to the calling routine.
      Begin
        Syecom[sys$l_options] = %c'R';
        Exec_image (Xfer_addr, Lstlun, AD ('R') );
        Return true ;
      End ;
    ! Call the device dependent module to produce a full report.
    Syecom[sys$l_options] = %c'S';
    Exec_image (Xfer_addr, Lstlun, AD ('S') );
    Return True;
  End;

[ EMBSC_SBIA,         ! SBI Adaptor error 13 %X0D
  EMBSC_CRD,          ! CRD log 14 %X0E
  EMBSC_EMM,          ! Environmental Monitor 15 %X0F
  EMBSC_HLT,          ! Processor Error Halt 16 %20
  EMBSC_CRBT ] :      ! Console Reboot 17 %X21
  Begin
    Exec_image (Xfer_addr);
  Return true;
```


: 1138
: 1139
: 1140
: 1141
: 1142
: 1143
: 11441695 2 End;
1696 2
1697 2 [3, 10, 18 to 31, 33, 43 to 63, 66 to 95, 102 to 111, outrange]:
1698 2 Return true;
1699 2 TES;
1700 2
1701 1 End;

.PSECT \$PLIT,NOWRT,NOEXE, PIC,2

00	00	00	52	000A8	P.AAR:	.ASCII	\R\<0><0><0>
				00000001	000AC	P.AAQ:	.LONG 1
				00000000	000B0		.ADDRESS P.AAR
00	00	00	53	000B4	P.AAT:	.ASCII	\S\<0><0><0>
				00000001	000B8	P.AAS:	.LONG 1
				00000000	000BC		.ADDRESS P.AAT
00	00	00	52	000C0	P.AAV:	.ASCII	\R\<0><0><0>
				00000001	000C4	P.AAU:	.LONG 1
				00000000	000C8		.ADDRESS P.AAV
00	00	00	53	000CC	P.AAX:	.ASCII	\S\<0><0><0>
				00000001	000D0	P.AAW:	.LONG 1
				00000000	000D4		.ADDRESS P.AAX

.PSECT \$OWNS,NOEXE, PIC,2

00014 XFER_ADDR:
 .BKLB 4
00018 CLASS: .BKLB 2
0001A TYPE: .BKLB 2

.PSECT \$CODE,NOWRT, PIC,2

03FC 00000 FULL_DISPATCHER:

59	00000000G	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9	: 1524	
58	00000000G	00	9E	00009	MOVAB	OPTION_FLAG, R9	:	
57	00000000G	00	9E	00010	MOVAB	WORST_ERROR, R8	:	
56	00000000G	00	9E	00017	MOVAB	PARSER_DATA, R7	:	
55	00000000G	00	9E	0001E	MOVAB	EMB+4, R6	:	
54	00000000G	00	9E	00025	MOVAB	P.AAQ, R5	:	
53	00000000G	00	9E	0002C	MOVAB	EXEC_IMAGE, R4	:	
52	00000000G	00	9E	00033	MOVAB	SYECOM+43, R3	:	
50		66	3C	0003A	MOVZWL	XFER_ADDR, R2	:	
01		50	B1	0003D	CMPL	EMB+4, R0	: 1564	
		0E	13	00040	BEQL	R0, #1	:	
0060	8F	50	B1	00042	CMPL	1\$:	
		07	13	00047	BEQL	R0, #96	: 1565	
0062	8F	50	B1	00049	CMPL	1\$:	
		0C	12	0004E	BEQL	R0, #98	: 1566	
06	A2	19	A6	9B	00050	2\$:	
04	A2	18	A6	9B	00055	MOVZBW	EMB+29, TYPE	: 1569
			07	11	0005A	MOVZBW	EMB+28, CLASS	: 1570
06	A2		50	B0	0005C	BRB	3\$: 1564
					MOVW	R0, TYPE	: 1574	

[illegible]

13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
9\$-5\$, -
9\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
13\$-5\$, -
6\$-5\$, -
6\$-5\$, -
6\$-5\$, -
7\$-5\$, -
7\$-5\$, -
7\$-5\$, -
13\$-5\$, -

.....

						13\$-5\$,-	
						13\$-5\$,-	
						13\$-5\$,-	
						13\$-5\$,-	
						13\$-5\$,-	
						13\$-5\$,-	
						13\$-5\$,-	
						13\$-5\$,-	
						13\$-5\$,-	
						6\$-5\$	
						13\$	
						PARSER_DATA, R0	1698
						(R0)	1616
						13\$	
						LSTLUN	
						R2	
						#2, EXEC_IMAGE	
						13\$	1692
						OPTION FLAG, R0	1628
						#14, (R0), 8\$	
						R1, #101	1629
						8\$	
						#82, SYECOM+43	1634
						R5	1636
						SYECOM	
						SYECOM+35	1635
						LSTLUN	
						R2	
						#5, EXEC_IMAGE	
						PARSER_DATA, R0	1643
						(R0)	
						13\$	
						#83, SYECOM+43	1646
						P.AAS	1648
						SYECOM	
						SYECOM+35	1647
						LSTLUN	
						R2	
						#5, EXEC_IMAGE	
						13\$	1692
						PARSER_DATA, R0	1667
						(R0)	
						10\$	
						OPTION FLAG, R0	1668
						#14, (R0), 10\$	
						#82, SYECOM+43	1674
						P.AAU	1675
						11\$	
						#83, SYECOM+43	1682
						P.AAW	1683
						LSTLUN	
						R2	
						#3, EXEC_IMAGE	
						13\$	1692
						R2	1693
						#1, EXEC_IMAGE	
						#1, R0	1694

ERF
V04-000

Errorlog Report Formatter

D 5
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMMASTER:[ERF.SRC]ERF.B32;1

Page 39
(12)

04 001E9

RET

; 1701

; Routine Size: 490 bytes, Routine Base: \$CODE + 0510

; 1145 1702 1

```
: 1147      1703 1 Routine OPEN_TEXT_LIB =
: 1148      1704 1
: 1149      1705 1 ++
: 1150      1706 1 Functional description
: 1151      1707 1
: 1152      1708 1 This routine set up the default library name then attempts
: 1153      1709 1 to translate ERFLIB. If there is no translation then the
: 1154      1710 1 default library name is used to open the text library.
: 1155      1711 1 If there is a translation then that string is used instead.
: 1156      1712 1 Once the library is opened, modules are read in and there
: 1157      1713 1 records parsed. These records are used to build the tables
: 1158      1714 1 which control device validation, device module secection,
: 1159      1715 1 and CPU validation. MODULE_NAME_DESC points at the name of
: 1160      1716 1 text module to be read and parsed. FUNCTION is a value which
: 1161      1717 1 specifies which record parser to use for a particular text
: 1162      1718 1 module.
: 1163      1719 1
: 1164      1720 1 Calling sequence
: 1165      1721 1
: 1166      1722 1 OPEN_TEXT_LIB ( )
: 1167      1723 1
: 1168      1724 1 Input parameters
: 1169      1725 1
: 1170      1726 1
: 1171      1727 1 Output parameters
: 1172      1728 1
: 1173      1729 1 None
: 1174      1730 1
: 1175      1731 1 Routine value
: 1176      1732 1
: 1177      1733 1 Worst error is returned.
: 1178      1734 1
: 1179      1735 1 ----
: 1180      1736 1
: 1181      1737 2 BEGIN
: 1182      1738 2
: 1183      1739 2 LOCAL
: 1184      1740 2 Buff:          $BBLOCK [80],
: 1185      1741 2 Desc:          VECTOR[2, LONG] INITIAL (80, buff),
: 1186      1742 2 Function,
: 1187      1743 2 Nocalled_needed: BYTE INITIAL (FALSE),
: 1188      1744 2 Status,
: 1189      1745 2 Text_library_name_desc,
: 1190      1746 2 Trnlmlst:      $itmlst_decl (items = 1);
: 1191      1747 2
: 1192      1748 2 Global
: 1193      1749 2 Ident;
: 1194      1750 2
: 1195      1751 2 Ident = $descriptor('V03-026');
: 1196      1752 2 Text_library_name_desc = $descriptor ('SYS$LIBRARY:ERFLIB.TLB') ;
: 1197      1753 2
: 1198      P 1754 2 $itmlst_init ( itmlst = trnlmlst, (itmcod = lnm$string, bufadr = .desc[1],
: 1199      1755 2 bufsiz = .desc[0], retlen = desc[0]));
: 1200      1756 2
: 1201      P 1757 2 If $trnlm ( attr = %ref(lnm$m case blind),
: 1202      P 1758 2 tabnam = lnm$file_dev_desc,
: 1203      P 1759 2 lognam = erflib_desc,
```



```
: 1204      1760      3      itmlst = trnlmlst)
: 1205      1761      2      Then
: 1206      1762      2      Text_library_name_desc = desc;
: 1207      1763      2
: 1208      1764      2      |
: 1209      1765      2      | Initialize text library control and open the library.
: 1210      1766      2      |
: 1211      1767      2      Status = LBR$INI_CONTROL ( Library_index, Library_func, Library_type ) ;
: 1212      1768      2      If NOT .status then Signal_stop (.status) ;
: 1213      1769      2
: 1214      1770      2      Status = LBR$OPEN ( Library_index, .Text_library_name_desc ) ;
: 1215      1771      2      If NOT .status then      ! Could not open library.
: 1216      1772      3      Begin
: 1217      1773      3      Signal_stop (msg$_searchfail, 1, .text_library_name_desc, .status) ;
: 1218      1774      3      End;
: 1219      1775      2
: 1220      1776      2      |
: 1221      1777      2      | Set to locate mode for reading records to parse.
: 1222      1778      2      |
: 1223      1779      2      |
: 1224      1780      2      CALL_FUNCTION ( LBR$SET_LOCATE (Library_index) );
: 1225      1781      2
: 1226      1782      2      |
: 1227      1783      2      |
: 1228      1784      2      | Sequence thru the reading and parsing of the text modules.
: 1229      1785      2      |
: 1230      1786      2      |
: 1231      1787      2      Incr loop_count from 1 to 11 do
: 1232      1788      3      Begin
: 1233      1789      3      Case .loop_count from 1 to 11 of set
: 1234      1790      3      [1]: ( Function = 1; Module_name_desc = $descriptor ('MAX_CLASS_SIZE') );
: 1235      1791      3      [2]: ( Function = 1; Module_name_desc = $descriptor ('CLASS_VALUES') );
: 1236      1792      3      [3]: ( Function = 3; Module_name_desc = $descriptor ('TABLE_SIZES') );
: 1237      1793      3      [4]:
: 1238      1794      4      Begin
: 1239      1795      4      Herald[msg$_msg_flg] = 1;      ! Message flages
: 1240      1796      4      Herald[msg$_arg_cnt] = 3;      ! Argument count
: 1241      1797      4      Herald[msg$_msg_id] = erf_herald;
: 1242      1798      4      Herald[msg$_new_flg] = 1;      ! New message flages
: 1243      1799      4      Herald[msg$_FA0_cnt] = 1;
: 1244      1800      4      Herald[msg$_FA0_arg1] = .ident;
: 1245      1801      4      $Putmsg (msgvec = herald);
: 1246      1802      4      Function = 4;
: 1247      1803      4      Module_name_desc = $descriptor ('DEVICES') ;
: 1248      1804      3      End;
: 1249      1805      3      [5]:
: 1250      1806      4      Begin
: 1251      1807      4      Function = 2;
: 1252      1808      4      Module_name_desc = $descriptor ('TRANSLATE_ENTRY_TABLE');
: 1253      1809      4      Table_address = .translate_entry_table;
: 1254      1810      4      Table_length = .max_misc_type;
: 1255      1811      4      Item_count = 0;
: 1256      1812      3      End;
: 1257      1813      3      [6]:
: 1258      1814      4      Begin
: 1259      1815      4      Function = 2;
: 1260      1816      4      Module_name_desc = $descriptor ('CPU_TYPES');
```

```
: 1261      1817  4      Table_address = .processor_type_table;
: 1262      1818  4      Table_length = .max_cpu_types;
: 1263      1819  4      Item_count = 0;
: 1264      1820  3      End;
: 1265      1821  3
: 1266      1822  3
: 1267      1823  4      [7]:
: 1268      1824  4      Begin
: 1269      1825  4      Function = 5;
: 1270      1826  4      Module_name_desc = $descriptor ('MIN_MODULE_NAMES');
: 1271      1827  4      Desc_table_address = .min_modules_desc;
: 1272      1828  4      Table_length = .max_cpu_types;
: 1273      1829  3      Item_count = 0;
: 1274      1830  3      End;
: 1275      1831  4      [8]:
: 1276      1832  4      Begin
: 1277      1833  4      Function = 5;
: 1278      1834  4      Module_name_desc = $descriptor ('MAX_MODULE_NAMES');
: 1279      1835  4      Desc_table_address = .max_modules_desc;
: 1280      1836  4      Table_length = .max_cpu_types;
: 1281      1837  3      Item_count = 0;
: 1282      1838  3      End;
: 1283      1839  3
: 1284      1840  3      [9]:
: 1285      1841  3      |
: 1286      1842  3      | THE NEXT THREE SECTIONS MUST BE DONE IN THIS SEQUENCE.
: 1287      1843  3      | This section loads the MIN_MAX_TABLE_SIZES table. Each
: 1288      1844  3      | table entry specifies the number of range pairs that
: 1289      1845  3      | exist for a particular CPU.
: 1290      1846  3      |
: 1291      1847  4      Begin
: 1292      1848  4      Function = 2;
: 1293      1849  4      Module_name_desc = $descriptor ('MIN_MAX_SIZES');
: 1294      1850  4      Table_address = .min_max_table_sizes;
: 1295      1851  4      Table_length = .max_cpu_types;
: 1296      1852  4      Item_count = 0;
: 1297      1853  3      End;
: 1298      1854  3
: 1299      1855  3      [10]:
: 1300      1856  3      |
: 1301      1857  3      | This section uses the contents of the MIN_MAX_TABLE_SIZES table
: 1302      1858  3      | to determine the size of the range tables. The base address of
: 1303      1859  3      | each range table is then saved.
: 1304      1860  3      |
: 1305      1861  4      Begin
: 1306      1862  4      Incr_range_loop from 1 to .max_cpu_types do
: 1307      1863  5      Begin
: 1308      1864  5      If .min_max_table_sizes[.range_loop] NEQ 0 then
: 1309      1865  6      Begin
: 1310      1866  6      Max_range_table_addr[.range_loop] =
: 1311      1867  6      get_vm ( (.min_max_table_sizes[.range_loop] + 1 ) * word_size);
: 1312      1868  6      Min_range_table_addr[.range_loop] =
: 1313      1869  6      get_vm ( (.min_max_table_sizes[.range_loop] + 1 ) * word_size);
: 1314      1870  6      End
: 1315      1871  5      Else
: 1316      1872  6      Begin
: 1317      1873  6      Max_range_table_addr[.range_loop] = 0;
```



```
: 1318      1874 6      Min_range_table_addr[.range_loop] = 0;
: 1319      1875 5      End;
: 1320      1876 4      End;
: 1321      1877 4
: 1322      1878 4
: 1323      1879 4      For each range table which has a non zero size, read a text library
: 1324      1880 4      module which will specify the min. ranges.
: 1325      1881 4
: 1326      1882 4      Incr range_loop from 1 to .max_cpu_types do
: 1327      1883 4      If .min_max_table_sizes[.range_loop] NEQ 0 then
: 1328      1884 5      Begin
: 1329      1885 5      Function = 2;
: 1330      1886 5      Module_name_desc = min_modules_desc[.range_loop,desc_one];
: 1331      1887 5      Table_address = .min_range_table_addr[.range_loop];
: 1332      1888 5      Table_length = .min_max_table_sizes[.range_loop];
: 1333      1889 5      Item_count = 0;
: 1334      1890 5      CALL_FUNCTION ( Get_library_text ( .Function, .Module_name_desc ));
: 1335      1891 4      End;
: 1336      1892 4      Nocall_needed = True;
: 1337      1893 3      End;
: 1338      1894 3
: 1339      1895 3
: 1340      1896 3      [11]:
: 1341      1897 3
: 1342      1898 3      For each range table which has a non zero size, read a text library
: 1343      1899 3      module which will specify the max. ranges.
: 1344      1900 3
: 1345      1901 4      Begin
: 1346      1902 4      Incr range_loop from 1 to .max_cpu_types do
: 1347      1903 4      If .min_max_table_sizes[.range_loop] NEQ 0 then
: 1348      1904 5      Begin
: 1349      1905 5      Function = 2;
: 1350      1906 5      Module_name_desc = max_modules_desc[.range_loop,desc_one];
: 1351      1907 5      Table_address = .max_range_table_addr[.range_loop];
: 1352      1908 5      Table_length = .min_max_table_sizes[.range_loop];
: 1353      1909 5      Item_count = 0;
: 1354      1910 5      CALL_FUNCTION ( Get_library_text ( .Function, .Module_name_desc ));
: 1355      1911 4      End;
: 1356      1912 4      Nocall_needed = True;
: 1357      1913 3      End;
: 1358      1914 3      TES;
: 1359      1915 3
: 1360      1916 3      If NOT .nocall_needed then      ! If nocall_needed is false then
: 1361      1917 4      CALL_FUNCTION ( Get_library_text ( .Function, .Module_name_desc ))
: 1362      1918 3      Else      ! else its true and reSet it to false.
: 1363      1919 3      Nocall_needed = false;
: 1364      1920 3
: 1365      1921 2      End;
: 1366      1922 2
: 1367      1923 2      Status = LBR$CLOSE ( Library_index );
: 1368      1924 2      If NOT .status then Signal_stop (.status) ;
: 1369      1925 2
: 1370      1926 2      Return true;
: 1371      1927 1      End;
```

```
.PSECT $PLIT,NOWRT,NOEXE, PIC,2

      36 32 30 2D 33 30 56 000D8 P.AAZ: .ASCII \V03-026\
      000DF 000E0 P.AAY: .BLKB 1
      00000007 000E4 P.ABB: .LONG 7
      00000000 000E8 P.ABB: .ADDRESS P.AAZ
      46 52 45 3A 59 52 41 52 42 49 4C 24 53 59 53 000F7 P.ABB: .ASCII \SYS$LIBRARY:ERFLIB.TLB\
      42 4C 54 2E 42 49 4C 000FE .BLKB 2
      00000016 00100 P.ABA: .LONG 22
      00000000 00104 P.ABA: .ADDRESS P.ABB
      45 5A 49 53 5F 53 53 41 4C 43 5F 58 41 4D 00108 P.ABD: .ASCII \MAX_CLASS_SIZE\
      00116 P.ABD: .BLKB 2
      0000000E 00118 P.ABC: .LONG 14
      00000000 0011C P.ABC: .ADDRESS P.ABD
      53 45 55 4C 41 56 5F 53 53 41 4C 43 00120 P.ABF: .ASCII \CLASS_VALUES\
      0000000C 0012C P.ABE: .LONG 12
      00000000 00130 P.ABE: .ADDRESS P.ABF
      53 45 5A 49 53 5F 45 4C 42 41 54 00134 P.ABH: .ASCII \TABLE_SIZES\
      0013F P.ABH: .BLKB 1
      0000000B 00140 P.ABG: .LONG 11
      00000000 00144 P.ABG: .ADDRESS P.ABH
      53 45 43 49 56 45 44 00148 P.ABJ: .ASCII \DEVICES\
      0014F P.ABJ: .BLKB 1
      00000007 00150 P.ABI: .LONG 7
      00000000 00154 P.ABI: .ADDRESS P.ABJ
      59 52 54 4E 45 5F 45 54 41 4C 53 4E 41 52 54 00158 P.ABL: .ASCII \TRANSLATE_ENTRY_TABLE\
      45 4C 42 41 54 5F 00167 P.ABL: .BLKB 3
      0016D P.ABL: .LONG 21
      00000015 00170 P.ABK: .ADDRESS P.ABL
      00000000 00174 P.ABK: .ASCII \CPU_TYPES\
      53 45 50 59 54 5F 55 50 43 00178 P.ABN: .BLKB 3
      00181 P.ABN: .LONG 9
      00000009 00184 P.ABM: .ADDRESS P.ABN
      00000000 00188 P.ABM: .ASCII \MIN_MODULE_NAMES\
      45 4D 41 4E 5F 45 4C 55 44 4F 4D 5F 4E 49 4D 0018C P.ABP: .LONG 16
      53 0019B P.ABP: .ADDRESS P.ABP
      00000010 0019C P.ABO: .ASCII \MAX_MODULE_NAMES\
      00000000 001A0 P.ABO: .LONG 16
      45 4D 41 4E 5F 45 4C 55 44 4F 4D 5F 58 41 4D 001A4 P.ABR: .ADDRESS P.ABR
      53 001B3 P.ABR: .LONG 16
      00000010 001B4 P.ABQ: .ADDRESS P.ABR
      00000000 001B8 P.ABQ: .ASCII \MIN_MAX_SIZES\
      53 45 5A 49 53 5F 58 41 4D 5F 4E 49 4D 001BC P.ABT: .BLKB 3
      001C9 P.ABT: .LONG 13
      0000000D 001CC P.ABS: .ADDRESS P.ABT
      00000000 001D0 P.ABS: .BLKB 4

.PSECT $GLOBAL$,NOEXE, PIC,2

000BC IDENT:: .EXTRN SYS$TRNLNM, SYS$PUTMSG

.PSECT $CODE,NOWRT, PIC,2

OFFC 00000 OPEN_TEXT_LIB:
```


[illegible]

					20\$-6\$,-			
					26\$-6\$,-			
	53		01	D0	000D3	7\$:	1790	
	69	38	AA	9E	000D6	MOVL		
			44	11	000DA	MOVAB		
	53		01	D0	000DC	BRB		
	69	4C	AA	9E	000DF	11\$	1791	
			3B	11	000E3	MOVL		
	53		03	D0	000E5	MOVAB		
	69	60	AA	9E	000E8	BRB		
			32	11	000EC	11\$	1792	
			8F	D0	000EE	MOVL		
B0	A9	00010003	8F	D0	000F6	MOVAB		
B4	A9	00000000G	8F	D0	000FE	BRB		
B8	A9	00010001	A9	D0	00106	11\$	1796	
BC	A9	54	7E	7C	0010B	MOVL		
			7E	D4	0010D	MOVAB		
			A9	9F	0010F	BRB		
		B0	04	FB	00112	11\$	1797	
			04	D0	00119	MOVAB		
00000000G	00		5C	11	00120	BRB		
	53		02	D0	00122	12\$:	1802	
	69	70	AA	9E	0011C	MOVL		
			CA	9E	00125	MOVAB		
	53		00	D0	0012A	BRB		
	69	0090	00	D0	0012A	19\$	1803	
			00	90	00132	MOVL		
28	A9	00000000G	3F	11	0013A	MOVAB		
2C	A9	00000000G	02	D0	0013C	BRB		
			CA	9E	0013F	12\$:	1807	
	53		2B	11	00149	MOVL		
	69	00A4	05	D0	0014B	MOVAB		
		08	CA	9E	0014E	BRB		
	53		A9	D0	00144	17\$	1815	
	69		05	D0	0014B	MOVL		
			CA	9E	0014E	MOVAB		
9C	A9	00BC	A9	D0	00153	BRB		
		F0	1C	11	00158	17\$	1816	
	53		05	D0	0015A	MOVL		
	69		CA	9E	0015D	MOVAB		
			A9	D0	00162	BRB		
9C	A9	00D4	0D	11	00167	17\$	1817	
		F4	02	D0	00169	16\$:	1818	
	53		CA	9E	0016C	MOVL		
	69	00EC	CA	9E	0016C	MOVAB		
			A9	D0	00171	BRB		
28	A9		A9	D0	00171	17\$:	1824	
2C	A9		A9	90	00176	MOVAB		
			CC	A9	D4	0017B	BRB	
			00D9	31	0017E	18\$:	1825	
	56		A9	3C	00181	20\$:	1826	
			52	D4	00185	MOVL		
			4A	11	00187	BRB		
	54		A9	D0	00189	21\$:	1832	
	50		A9	D0	0018D	MOVL		
			6042	3C	00191	MOVAB		
	50		32	13	00195	BRB		
			01	78	00197	22\$	1833	
7E	50		02	C0	0019B	MOVL		
	6E		01	FB	0019E	MOVAB		
			50	D0	001A5	BRB		
00000000G	00		A9	D0	001A9	17\$:	1834	
	6442					MOVZWL		
	54	FC				CLRL		
						BRB		
						23\$	1835	
						MAX_RANGE_TABLE_ADDR, R4	1866	
						MIN_MAX_TABLE_SIZES, R0	1864	
						(R0)[RANGE_LOOP], R0		
						BEQL		
						22\$		
						#1, R0, -(SP)	1867	
						#2, (SP)		
						#1, GET VM		
						R0, (R4)[RANGE_LOOP]		
						MIN_RANGE_TABLE_ADDR, R4	1868	

	50	F8	A9	D0	001AD	MOVL	MIN MAX TABLE SIZES, R0	1869
	50		6042	3C	001B1	MOVZWL	(R0)[RANGE_LOOP], R0	
7E	50		01	78	001B5	ASHL	#1, R0, -(SP)	
	6E		02	C0	001B9	ADDL2	#2, (SP)	
00000000G	00		01	FB	001BC	CALLS	#1, GET_VM	
	6442		50	D0	001C3	MOVL	R0, (R4)[RANGE_LOOP]	
			0A	11	001C7	BRB	23\$	1864
			6442	D4	001C9	CLRL	(R4)[RANGE_LOOP]	1873
	50	FC	A9	D0	001CC	MOVL	MIN RANGE TABLE_ADDR, R0	1874
			6042	D4	001D0	CLRL	(R0)[RANGE_LOOP]	
B2	52		56	F3	001D3	AOBLEQ	R6, RANGE_LOOP, 21\$	1862
	52	E2	A9	3C	001D7	MOVZWL	MAX CPU TYPES, R2	1882
			54	D4	001DB	CLRL	RANGE_LOOP	
			33	11	001DD	BRB	25\$	
	51	F8	A9	D0	001DF	MOVL	MIN MAX TABLE SIZES, R1	1883
			6144	B5	001E3	TSTW	(R1)[RANGE_LOOP]	
			2A	13	001E6	BEQL	25\$	
	53		02	D0	001E8	MOVL	#2, FUNCTION	1885
	50	F0	A9	D0	001EB	MOVL	MIN MODULES_DESC, R0	1886
	69		6044	7E	001EF	MOVAQ	(R0)[RANGE_LOOP], MODULE_NAME_DESC	
	50	FC	A9	D0	001F3	MOVL	MIN RANGE TABLE_ADDR, R0	1887
	28		6044	D0	001F7	MOVL	(R0)[RANGE_LOOP], TABLE_ADDRESS	
2C	A9		6144	33	001FC	CVTWB	(R1)[RANGE_LOOP], TABLE_LENGTH	1888
		CC	A9	D4	00201	CLRL	ITEM COUNT	1889
			69	DD	00204	PUSHL	MODULE_NAME_DESC	1890
			53	DD	00206	PUSHL	FUNCTION	
00000000V	00		02	FB	00208	CALLS	#2, GET_LIBRARY_TEXT	
	7A		50	E9	0020F	BLBC	STATUS, -34\$	
C9	54		52	F3	00212	AOBLEQ	R2, RANGE_LOOP, 24\$	1883
			3F	11	00216	BRB	29\$	1892
	52	E2	A9	3C	00218	MOVZWL	MAX CPU TYPES, R2	1902
			54	D4	0021C	CLRL	RANGE_LOOP	
			33	11	0021E	BRB	28\$	
	51	F8	A9	D0	00220	MOVL	MIN MAX TABLE SIZES, R1	1903
			6144	B5	00224	TSTW	(R1)[RANGE_LOOP]	
			2A	13	00227	BEQL	28\$	
	53		02	D0	00229	MOVL	#2, FUNCTION	1905
	50	F4	A9	D0	0022C	MOVL	MAX MODULES_DESC, R0	1906
	69		6044	7E	00230	MOVAQ	(R0)[RANGE_LOOP], MODULE_NAME_DESC	
	50	E8	A9	D0	00234	MOVL	MAX RANGE TABLE_ADDR, R0	1907
	28		6044	D0	00238	MOVL	(R0)[RANGE_LOOP], TABLE_ADDRESS	
2C	A9		6144	33	0023D	CVTWB	(R1)[RANGE_LOOP], TABLE_LENGTH	1908
		CC	A9	D4	00242	CLRL	ITEM COUNT	1909
			69	DD	00245	PUSHL	MODULE_NAME_DESC	1910
			53	DD	00247	PUSHL	FUNCTION	
00000000V	00		02	FB	00249	CALLS	#2, GET_LIBRARY_TEXT	
	39		50	E9	00250	BLBC	STATUS, -34\$	
C9	54		52	F3	00253	AOBLEQ	R2, RANGE_LOOP, 27\$	1903
	58		01	90	00257	MOVB	#1, NOCALL_NEEDED	1912
	0F		58	E8	0025A	BLBS	NOCALL_NEEDED, 31\$	1916
			69	DD	0025D	PUSHL	MODULE_NAME_DESC	1917
			53	DD	0025F	PUSHL	FUNCTION	
00000000V	00		02	FB	00261	CALLS	#2, GET_LIBRARY_TEXT	
	03		50	E8	00268	BLBS	STATUS, -32\$	
				04	0026B	RET		
			58	94	0026C	CLRB	NOCALL_NEEDED	1919
FE45	55		0B	F1	0026E	ACBL	#11, #T, LOOP_COUNT, 5\$	1787

ERF
V04-000

Errorlog Report Formatter

M 5
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 48
(13)

00000000G	00	D4	A9	9F	00274	PUSHAB	LIBRARY INDEX	:	1923
	57		01	FB	00277	CALLS	#1, LBR\$CLOSE	:	
	05		50	D0	0027E	MOVL	R0, STATUS	:	
			57	E8	00281	BLBS	STATUS, 33\$:	1924
			57	DD	00284	PUSHL	STATUS	:	
	68		01	FB	00286	CALLS	#1, LIB\$STOP	:	
	50		01	D0	00289	MOVL	#1, R0	:	1926
			04	0028C	34\$:	RET		:	1927

; Routine Size: 653 bytes, Routine Base: \$CODE + 06FA


```
1373 1928 1 Routine GET_LIBRARY_TEXT ( Function, module_name ) =
1374 1929 2 BEGIN
1375 1930 3 ++
1376 1931 4 Functional description
1377 1932 5
1378 1933 6     This routine looks up the text module name specified. It
1379 1934 7     reads the records from that text module. If a record
1380 1935 8     does not have a comment character in the first three
1381 1936 9     character positions and the record is not of length zero,
1382 1937 10    then the parsing routine specified by FUNCTION is called.
1383 1938 11
1384 1939 12 Calling sequence
1385 1940 13
1386 1941 14     GET_LIBRARY_TEXT ( Function, module_name )
1387 1942 15
1388 1943 16 Input parameters
1389 1944 17
1390 1945 18     Function : Value specifying;
1391 1946 19         1 Build class tables
1392 1947 20         2 Parse and convert to binary a list of values
1393 1948 21         3 Allocate and initialize processor and device tables
1394 1949 22         4 Parse device description records. See text module
1395 1950 23         DEVICES for more information.
1396 1951 24
1397 1952 25     Module_name : Address of descriptor for module name.
1398 1953 26
1399 1954 27 Output parameters
1400 1955 28
1401 1956 29     None
1402 1957 30
1403 1958 31 Routine value
1404 1959 32
1405 1960 33     Worst error is returned.
1406 1961 34
1407 1962 35 ----
1408 1963 36
1409 1964 37 LOCAL
1410 1965 38     Offset,
1411 1966 39     Position,
1412 1967 40     Status;
1413 1968 41
1414 1969 42
1415 1970 43 Use MODULE_NAME as the key to find the text module in library.
1416 1971 44
1417 1972 45
1418 1973 46 Status = LBR$LOOKUP_KEY ( Library_index, Module_name, Text_rfa ) ;
1419 1974 47 If NOT .status then Signal (erf_badmodnam, 1, .module_name, .status) ;
1420 1975 48
1421 1976 49
1422 1977 50
1423 1978 51 READ A RECORD FROM THE TEXT LIBRARY
1424 1979 52 If the record length is not zero then case to a decode routine.
1425 1980 53 Search the record for the comment character '!'.
1426 1981 54 If "!" is in one of the first three positions get a new record.
1427 1982 55
1428 1983 56
1429 1984 57 While Status = LBR$GET_RECORD ( Library_index, Record_desc, Record_desc) do
```

```

1430      1985 3      Begin
1431      1986 3      If .Record_desc [dsc$w_length] NEQ 0 then
1432      1987 4          Begin
1433      1988 4              Position = CH$FIND_CH (.Record_desc [dsc$w_length],
1434      1989 4                  .Record_desc [dsc$a_pointer], %c'!' );
1435      1990 4              If .position NEQ 0 then
1436      1991 4                  Offset = CH$DIFF ( .position , .Record_desc [dsc$a_pointer])
1437      1992 4              Else
1438      1993 4                  Offset = 4;
1439      1994 4              If .Offset GTR 3 then
1440      1995 4                  Case .function from 1 to 5 of set
1441      1996 4                      [1]: Build_class_tables ();           ! Inits device class table
1442      1997 4                      [2]: Parse_max_min_table_record ();    ! Min max table parser
1443      1998 4                      [3]: Parse_max_table_size ();         ! Parse & convert to binary
1444      1999 4                      [4]: Parse_device_desc_record ();     ! Parse & return strings
1445      2000 4                      [5]: Parse_module_names ();
1446      2001 4                  Tes;
1447      2002 4              End;
1448      2003 3          End;
1449      2004 2      End;
1450      2005 2
1451      2006 2      Item_count = 0;                                     ! Global used by several routines
1452      2007 2                                                         ! as there array index.
1453      2008 2      Return true ;
1454      2009 1      End ;

```

		007C 00000 GET_LIBRARY TEXT:					
	56	00000000'	00	9E	00002	.WORD Save R2,R3,R4,R5,R6	: 1928
		20	A6	9F	00009	MOVAB RECORD_DESC, R6	
		08	AC	DD	0000C	PUSHAB TEXT_RFA	: 1973
		C4	A6	9F	0000F	PUSHL MODULE_NAME	
00000000G	00		03	FB	00012	PUSHAB LIBRARY_INDEX	
	55		50	D0	00019	CALLS #3, LBR\$LOOKUP_KEY	
	14		55	E8	0001C	MOVL R0, STATUS	
			55	DD	0001F	BLBS STATUS, 1\$: 1974
		08	AC	DD	00021	PUSHL STATUS	
			01	DD	00024	PUSHL MODULE_NAME	
		00000000G	8F	DD	00026	PUSHL #1	
00000000G	00		04	FB	0002C	PUSHL #ERF_BADMODNAM	
			56	DD	00033	CALLS #4, CIB\$SIGNAL	
			56	DD	00035	PUSHL R6	: 1984
		C4	A6	9F	00037	PUSHL R6	
00000000G	00		03	FB	0003A	PUSHAB LIBRARY_INDEX	
	55		50	D0	00041	CALLS #3, LBR\$GET_RECORD	
	60		55	E9	00044	MOVL R0, STATUS	
	50		66	3C	00047	BLBC STATUS, 11\$	
			E7	13	0004A	MOVZWL RECORD_DESC, R0	: 1986
	52	04	A6	D0	0004C	BEQL 1\$	
62	50		21	3A	00050	MOVL RECORD_DESC+4, R2	: 1989
			02	12	00054	LOCC #33, R0, (R2)	: 1988
			51	D4	00056	BNEQ 2\$	
			51	D0	00058	CLRL R1	
	54		51	D0	00058	MOVL R1, POSITION	

ERF
V04-000

Errorlog Report Formatter

C 6
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 51
(14)

53	54	06 13 0005B	BEQL	3\$:	1990
		52 C3 0005D	SUBL3	R2, POSITION, OFFSET	:	1991
	53	03 11 00061	BRB	4\$:	
	03	04 D0 00063 3\$:	MOVL	#4, OFFSET	:	1993
		53 D1 00066 4\$:	CMPL	OFFSET, #3	:	1995
		C8 15 00069	BLEQ	1\$:	
0025	04 01	04 AC CF 0006B	CASEL	FUNCTION, #1, #4	:	1996
	001C 0013	000A 00070 5\$:	.WORD	6\$-5\$,-	:	
		002E 00078		7\$-5\$,-	:	
				8\$-5\$,-	:	
				9\$-5\$,-	:	
				10\$-5\$:	
	00000000V 00	00 FB 0007A 6\$:	CALLS	#0, BUILD_CLASS_TABLES	:	1997
		B0 11 00081	BRB	1\$:	
	00000000V 00	00 FB 00083 7\$:	CALLS	#0, PARSE_MAX_MIN_TABLE_RECORD	:	1998
		A7 11 0008A	BRB	1\$:	
	00000000V 00	00 FB 0008C 8\$:	CALLS	#0, PARSE_MAX_TABLE_SIZE	:	1999
		9E 11 00093	BRB	1\$:	
	00000000V 00	00 FB 00095 9\$:	CALLS	#0, PARSE_DEVICE_DESC_RECORD	:	2000
		95 11 0009C	BRB	1\$:	
	00000000V 00	00 FB 0009E 10\$:	CALLS	#0, PARSE_MODULE_NAMES	:	2001
		8C 11 000A5	BRB	1\$:	1996
		BC A6 D4 000A7 11\$:	CLRL	ITEM_COUNT	:	2006
	50	01 D0 000AA	MOVL	#1, R0	:	2008
		04 000AD	RET		:	2009

; Routine Size: 174 bytes, Routine Base: \$CODE + 0987

; 1455 2010 1

```
: 1457      2011 1 Routine BUILD_CLASS_TABLES =
: 1458      2012 2 BEGIN
: 1459      2013 2 ++
: 1460      2014 2 Functional description
: 1461      2015 2
: 1462      2016 2     This routine allocates memory for tables which will be filled
: 1463      2017 2     with the information obtained from the ERF text library in
: 1464      2018 2     SYSS$LIBRARY. See the text library modules for a description
: 1465      2019 2     of the text library records.
: 1466      2020 2
: 1467      2021 2 Calling sequence
: 1468      2022 2
: 1469      2023 2     Build_class_tables ()
: 1470      2024 2
: 1471      2025 2 Input parameters
: 1472      2026 2
: 1473      2027 2
: 1474      2028 2 Output parameters
: 1475      2029 2
: 1476      2030 2     Class_dir, class_names, dev_addrs_ptr, dev_class_ptr
: 1477      2031 2
: 1478      2032 2 Routine value
: 1479      2033 2
: 1480      2034 2     Worst error is returned.
: 1481      2035 2
: 1482      2036 2 ----
: 1483      2037 2
: 1484      2038 2 OWN
: 1485      2039 2     Context,           ! Continuation flag if this flag is set then more
: 1486      2040 2                               ! values in comma seperated list
: 1487      2041 2     Size,           ! Length of the field pointed to by VALUE_ADDR
: 1488      2042 2     Status,        ! Status after a convert
: 1489      2043 2     Index,         ! The first field of a text record
: 1490      2044 2     Value_addr;      ! Pointer to current filed in the text record
: 1491      2045 2
: 1492      2046 2 !LOCAL
: 1493      2047 2     Class_names;          ! Address of DC$_xxx strings
: 1494      2048 2 !Class_names = Get_vm ((.size+1) * 13); ! Class name + string size = 13
```



```

: 1496      2049 2 |
: 1497      2050 2 | Call to obtain the index and the current value address (value_addr).
: 1498      2051 2 |
: 1499      2052 2 |
: 1500      2053 2 Context = 0;
: 1501      2054 2 CALL_FUNCTION ( Parse_text_record ( context, index, value_addr, size ) );
: 1502      2055 2 |
: 1503      2056 2 |
: 1504      2057 2 |
: 1505      2058 2 |
: 1506      2059 2 | Convert the current value from ASCII to binary and use it to allocate
: 1507      2060 2 | memory for tables. Use the index value to determin which table to build.
: 1508      2061 2 |
: 1509      2062 2 Status = LIB$CVT_DTB ( .size, .Value_addr, size );
: 1510      2063 2 If NOT .status then Signal (erf_cvterr, 2,.size,value_addr) ;
: 1511      2064 2 |
: 1512      2065 2 If .class_dir EQL 0 then
: 1513      2066 2 Begin
: 1514      2067 2 Class_dir = Get_vm ((.size+1) * 6);      ! Device Class(2 bytes + device name table addr (longword)= 6
: 1515      2068 2 Max_classes = .size;                  ! Total number of dev. classes
: 1516      2069 2 Dev_addrs_ptr = Class_dir[.max_classes+1];
: 1517      2070 2 Dev_class_ptr = .class_dir ;
: 1518      2071 2 End
: 1519      2072 2 Else
: 1520      2073 2 Begin
: 1521      2074 2 If .index GTR .max_classes then signal (erf_toomancls, 1, .index);
: 1522      2075 2 Class_dir[.index] = .size; ! Set device class values
: 1523      2076 2 End;
: 1524      2077 2 |
: 1525      2078 2 Return true ;
: 1526      2079 1 End ;
```

.PSECT \$OWNS,NOEXE, PIC,2

```
0001C CONTEXT: .BLKB 4
00020 SIZE: .BLKB 4
00024 STATUS: .BLKB 4
00028 INDEX: .BLKB 4
0002C VALUE_ADDR:
.BLKB 4
```

.PSECT \$CODE,NOWRT, PIC,2

003C 00000 BUILD_CLASS TABLES:

```

55 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5
54 00000000G 00 9E 00009 MOVAB LIB$SIGNAL, R5
53 00000000G 00 9E 00010 MOVAB CLASS_DIR, R4
52 00000000G 00 9E 00017 MOVAB MAX_CLASSES, R3
FC A2 D4 0001E MOVAB SIZE, R2
52 DD 00021 CLRL CONTEXT
OC A2 9F 00023 PUSHL R2
08 A2 9F 00026 PUSHAB VALUE_ADDR
PUSHAB INDEX
```

: 2011

: 2053
: 2054

00000000V	00	FC	A2	9F	00029	PUSHAB	CONTEXT	:
	76		04	FB	0002C	CALLS	#4, PARSE_TEXT_RECORD	:
			50	E9	00033	BLBC	STATUS, 5\$:
			52	DD	00036	PUSHL	R2	2062
		0C	A2	DD	00038	PUSHL	VALUE_ADDR	:
			62	DD	0003B	PUSHL	SIZE	:
00000000G	00		03	FB	0003D	CALLS	#3, LIB\$CVT_DTB	:
	04		50	D0	00044	MOVL	R0, STATUS	:
		04	A2	E8	00048	BLBS	STATUS, 1\$	2063
		0C	A2	9F	0004C	PUSHAB	VALUE_ADDR	:
			62	DD	0004F	PUSHL	SIZE	:
			02	DD	00051	PUSHL	#2	:
		00000000G	8F	DD	00053	PUSHL	#ERF CVTERR	:
	65		04	FB	00059	CALLS	#4, [IB\$SIGNAL	:
			64	D5	0005C	TSTL	CLASS_DIR	2065
			26	12	0005E	BNEQ	2\$:
50			06	C5	00060	MULL3	#6, SIZE, R0	2067
		06	A0	9F	00064	PUSHAB	6(R0)	:
			01	FB	00067	CALLS	#1, GET_VM	:
00000000G	00		50	D0	0006E	MOVL	R0, CLASS_DIR	:
	64		62	90	00071	MOVB	SIZE, MAX_CLASSES	2068
	63		64	D0	00074	MOVL	CLASS_DIR, R1	2069
	51		63	9A	00077	MOVZBL	MAX_CLASSES, R0	:
	50		51	D0	00080	MOVW	2(RT)[R0], DEV_ADDRS_PTR	:
BF	A3	02	A140	3E	0007A	MOVL	R1, DEV_CLASS_PTR	2070
C3	A3		23	11	00084	BRB	4\$	2065
			A2	D0	00086	MOVL	INDEX, R0	2074
50		08	00	ED	0008A	CMPZV	#0, #8, MAX_CLASSES, R0	:
63			0D	18	0008F	BGEQ	3\$:
			50	DD	00091	PUSHL	R0	:
			01	DD	00093	PUSHL	#1	:
		00000000G	8F	DD	00095	PUSHL	#ERF TOOMANCLS	:
	65		03	FB	0009B	CALLS	#3, [IB\$SIGNAL	:
	51		64	D0	0009E	MOVL	CLASS_DIR, R1	2075
	50		A2	D0	000A1	MOVL	INDEX, R0	:
6140		08	62	B0	000A5	MOVW	SIZE, (R1)[R0]	:
50			01	D0	000A9	MOVL	#1, R0	2078
			04	000AC	5\$:	RET		2079

; Routine Size: 173 bytes, Routine Base: \$CODE + 0A35


```
: 1528      2080 1 Routine PARSE_MAX_MIN_TABLE_RECORD =
: 1529      2081 2 BEGIN
: 1530      2082 2 ++
: 1531      2083 2 Functional description
: 1532      2084 2
: 1533      2085 2     This routine calls parse_text_record to obtain a value from
: 1534      2086 2     the comma seperated list of values. The value is converted
: 1535      2087 2     to binary and place in a table.
: 1536      2088 2
: 1537      2089 2 Calling sequence
: 1538      2090 2
: 1539      2091 2 Input parameters
: 1540      2092 2
: 1541      2093 2
: 1542      2094 2 Output parameters
: 1543      2095 2
: 1544      2096 2     Fills the table specified by TABLE_ADDRESS.
: 1545      2097 2
: 1546      2098 2 Routine value
: 1547      2099 2
: 1548      2100 2     Worst error is returned.
: 1549      2101 2
: 1550      2102 2 ----
: 1551      2103 2 Local
: 1552      2104 2     Context,
: 1553      2105 2     Index,
: 1554      2106 2     Size,
: 1555      2107 2     Status,
: 1556      2108 2     Value_addr;
: 1557      2109 2
```

2080

2110
2114
2116

2117

2119

2122

2123

ERF
V04-000

Errorlog Report Formatter

1 6
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 57
(18)

	0C	AE	DD	00064	PUSHL	SIZE	:
		02	DD	00067	PUSHL	#2	:
	00000000G	8F	DD	00069	PUSHL	#ERF_CVTERR	:
64		04	FB	0006F	CALLS	#4, [IBSSIGNAL	:
51	5C	A3	D0	00072	3\$:	MOVL	TABLE_ADDRESS, R1
50		63	D0	00076	MOVL	ITEM_COUNT, R0	2125
6140	08	AE	B0	00079	MOVW	SIZE, (R1)[R0]	:
01	04	AE	D1	0007E	CMPL	CONTEXT, #1	2129
		92	13	00082	BEQL	1\$:
50		01	D0	00084	4\$:	MOVL	#1, R0
		04	00087	5\$:	RET		2131
							2132

; Routine Size: 136 bytes, Routine Base: \$CODE + 0AE2

```
1583 2133 1 Routine PARSE_MAX_TABLE_SIZE =
1584 2134 2 BEGIN
1585 2135 2 ++
1586 2136 2 Functional description
1587 2137 2
1588 2138 2 With the information that is returned from a call to
1589 2139 2 'PARSE_TEXT_RECORD', this routine allocates storage.
1590 2140 2 The index value of the library record being parsed
1591 2141 2 determines which table pointers are initialized.
1592 2142 2 It also sets up the table of addresses pointed to
1593 2143 2 by 'DEV_ADDR_PTR'.
1594 2144 2
1595 2145 2 Calling sequence
1596 2146 2
1597 2147 2 Input parameters
1598 2148 2
1599 2149 2 None.
1600 2150 2
1601 2151 2 Output parameters
1602 2152 2
1603 2153 2 All the table pointers for devices and cpu tables
1604 2154 2 are set up here and are global.
1605 2155 2
1606 2156 2 Routine value
1607 2157 2
1608 2158 2 Worst error is returned.
1609 2159 2
1610 2160 2 ----
1611 2161 2
1612 2162 2 OWN
1613 2163 2 Dc_class: word,      ! Temp for device class
1614 2164 2 Device_addr,        ! Temp for device table address
1615 2165 2 Context,            ! Flag which specifies if there are more items in the text record
1616 2166 2 Image_addr,
1617 2167 2 Index,              ! Value of the first item in the text record
1618 2168 2 Size,               ! Size of the item returned.
1619 2169 2 Status,             ! Status of the LIB$ call
1620 2170 2 Value_addr,         ! Address of the current value in the text record
1621 2171 2 Version_addr,
1622 2172 2 Xfer_addr;
1623 2173 2
```



```
: 1625      2174 2 |
: 1626      2175 2 | Call to obtain the index and the current value address (value_addr).
: 1627      2176 2 |
: 1628      2177 2 | Context = 0;
: 1629      2178 2 | CALL_FUNCTION ( Parse_text_record ( context, index, value_addr, size ) );
: 1630      2179 2 |
: 1631      2180 2 |
: 1632      2181 2 | Convert the current value from ASCII to binary and use it to allocate
: 1633      2182 2 | memory for tables. Use the index value to determin which table to build.
: 1634      2183 2 |
: 1635      2184 2 | Status = LIB$CVT_DTB ( .size, .Value_addr, size );
: 1636      2185 2 | If NOT .status then Signal (erf_cvterr, 2,.size,value_addr) ;
: 1637      2186 2 |
: 1638      2187 2 |
: 1639      2188 2 | Update the size for non-used first locations in tables.
: 1640      2189 2 |
: 1641      2190 2 | Size = .size + 1 ;
: 1642      2191 2 |
: 1643      2192 2 |
: 1644      2193 2 | If INDEX is equal to one then allocate storage for the
: 1645      2194 2 | CPU and verification table and save the table addresses.
: 1646      2195 2 |
: 1647      2196 2 | If .index EQL 1 then
: 1648      2197 2 | Begin
: 1649      2198 2 | Local Amount;
: 1650      2199 2 | Amount = .size * longword;
: 1651      2200 2 | Max_cpu_types = .size -1;
: 1652      2201 2 | Min_range_table_addr = get_vm (.amount);
: 1653      2202 2 | Max_range_table_addr = get_vm (.amount);
: 1654      2203 2 | Min_modules_desc = get_vm (.amount*2);
: 1655      2204 2 | Max_modules_desc = get_vm (.amount*2);
: 1656      2205 2 | Processor_type_table = get_vm (.size * word_size);
: 1657      2206 2 | Min_max_table_sizes = get_vm (.size * word_size);
: 1658      2207 2 | Return true;
: 1659      2208 2 | End;
: 1660      2209 2 |
: 1661      2210 2 |
: 1662      2211 2 | Allocate the storage for the device, version number, xfer address,
: 1663      2212 2 | and image name tables.
: 1664      2213 2 |
: 1665      2214 2 | Device_addr = get_vm (.size * word_size);
: 1666      2215 2 | Version_addr = get_vm (.size * word_size);
: 1667      2216 2 | Xfer_addr = get_vm (.size * longword);
: 1668      2217 2 | Image_addr = get_vm (.size * descriptor_length);
: 1669      2218 2 |
: 1670      2219 2 |
: 1671      2220 2 |
: 1672      2221 2 | Via the index determine which entry is being processed and
: 1673      2222 2 | copy the addresses/size of the tables to the appropriate places.
: 1674      2223 2 |
: 1675      2224 2 | Case .index from 2 to 9 of set
: 1676      2225 2 | [2]: Begin
: 1677      2226 2 | Disk_devices = .device_addr;
: 1678      2227 2 | Disk_version = .version_addr;
: 1679      2228 2 | Disk_xfer_addr = .xfer_addr;
: 1680      2229 2 | Disk_image = .image_addr;
: 1681      2230 2 | Max_disk_type = .size - 1;
```

```
: 1682      2231      3      Disk_devices[0] = .size;
: 1683      2232      3      Dc_class = DC$_DISK;
: 1684      2233      3      End;
: 1685      2234      3
: 1686      2235      3      [3]: Begin
: 1687      2236      3      Tape_devices = .device_addr;
: 1688      2237      3      Tape_version = .version_addr;
: 1689      2238      3      Tape_xfer_addr = .xfer_addr;
: 1690      2239      3      Tape_image = .image_addr;
: 1691      2240      3      Max_tape_type = .size - 1;
: 1692      2241      3      Tape_devices[0] = .size;
: 1693      2242      3      Dc_class = DC$_TAPE;
: 1694      2243      3      End;
: 1695      2244      3
: 1696      2245      3      [4]: Begin
: 1697      2246      3      Scm_devices = .device_addr;
: 1698      2247      3      Scm_version = .version_addr;
: 1699      2248      3      Scm_xfer_addr = .xfer_addr;
: 1700      2249      3      Scm_image = .image_addr;
: 1701      2250      3      Max_scm_type = .size - 1;
: 1702      2251      3      Scm_devices[0] = .size;
: 1703      2252      3      Dc_class = DC$_SCM;
: 1704      2253      3      End;
: 1705      2254      3
: 1706      2255      3      [5]: Begin
: 1707      2256      3      Lp_devices = .device_addr;
: 1708      2257      3      Lp_version = .version_addr;
: 1709      2258      3      Lp_xfer_addr = .xfer_addr;
: 1710      2259      3      Lp_image = .image_addr;
: 1711      2260      3      Max_lp_type = .size - 1;
: 1712      2261      3      Lp_devices[0] = .size;
: 1713      2262      3      Dc_class = DC$_LP;
: 1714      2263      3      End;
: 1715      2264      3
: 1716      2265      3      [6]: Begin
: 1717      2266      3      Realtime_devices = .device_addr;
: 1718      2267      3      Realtime_version = .version_addr;
: 1719      2268      3      Realtime_xfer_addr = .xfer_addr;
: 1720      2269      3      Realtime_image = .image_addr;
: 1721      2270      3      Max_realtime_type = .size - 1;
: 1722      2271      3      Realtime_devices[0] = .size;
: 1723      2272      3      Dc_class = DC$_REALTIME;
: 1724      2273      3      End;
: 1725      2274      3
: 1726      2275      3      [7]: Begin
: 1727      2276      3      Bus_devices = .device_addr;
: 1728      2277      3      Bus_version = .version_addr;
: 1729      2278      3      Bus_xfer_addr = .xfer_addr;
: 1730      2279      3      Bus_image = .image_addr;
: 1731      2280      3      Max_bus_type = .size - 1;
: 1732      2281      3      Bus_devices[0] = .size;
: 1733      2282      3      Dc_class = DC$_BUS;
: 1734      2283      3      End;
: 1735      2284      3
: 1736      2285      3      [8]: Begin
: 1737      2286      3      Packet_processor_devices = .device_addr;
: 1738      2287      3      Packet_processor_version = .version_addr;
```



```
: 1739      2288      3      Packet_processor_xfer_addr = .xfer_addr;
: 1740      2289      3      Packet_processor_image = .image_addr;
: 1741      2290      3      Max_misc_type = .size - 1;
: 1742      2291      3      Packet_processor_devices[0] = .size;
: 1743      2292      3      Translate_entry_table = get_vm (.size * word_size);
: 1744      2293      3      Dc_class = DCS_ZERO_CLASS;
: 1745      2294      3      End;
: 1746      2295      3
: 1747      2296      3      [9]: Begin
: 1748      2297      3      Workstation_devices = .device_addr;
: 1749      2298      3      Workstation_version = .version_addr;
: 1750      2299      3      Workstation_xfer_addr = .xfer_addr;
: 1751      2300      3      Workstation_image = .image_addr;
: 1752      2301      3      Max_Workstation_type = .size - 1;
: 1753      2302      3      Workstation_devices[0] = .size;
: 1754      2303      3      Dc_class = DCS_WORKSTATION;
: 1755      2304      3      End;
: 1756      2305      3
: 1757      2306      3      [OUTRANGE]: Begin
: 1758      2307      3      Signal (erf_badevval, 1, .index, .module_name_desc) ;
: 1759      2308      3      Return true ;
: 1760      2309      3      End;
: 1761      2310      3
: 1762      2311      3      TES;
: 1763      2312      3
: 1764      2313      3      !
: 1765      2314      3      ! Fill in the device class address of the 'class_dir' table. It
: 1766      2315      3      ! contains the pointers to the device class specific tables (devices,
: 1767      2316      3      ! version number, xfer address, and image name).
: 1768      2317      3      !
: 1769      2318      3      ! Incr count from 1 to .max_classes do
: 1770      2319      3      ! Begin
: 1771      2320      3      ! If .dev_class_ptr[.count] EQL .dc_class then      ! Make sure its the right slot
: 1772      2321      4      !     Begin      ! for the address.
: 1773      2322      4      !     Dev_addrs_ptr[.count] = .device_addr;      ! Save the address of the
: 1774      2323      4      !     Return true ;      ! device name tables.
: 1775      2324      3      !     End;
: 1776      2325      3      ! End;
: 1777      2326      3
: 1778      2327      2      Signal (erf_clstblerr, 1,.dc_class) ;
: 1779      2328      2      Return true;
: 1780      2329      1      End ;
```

.PSECT \$OWNS,NOEXE, PIC,2

```
00030 DC_CLASS:
00032      .BLKB      2
00034 DEVICE_ADDR:
00038      .BLKB      4
00038 CONTEXT: .BLKB      4
0003C IMAGE_ADDR:
00040      .BLKB      4
00040 INDEX:   .BLKB      4
00044 SIZE:   .BLKB      4
```

00048 STATUS: .BLKB 4
0004C VALUE_ADDR: .BLKB 4
00050 VERSION_ADDR: .BLKB 4
00054 XFER_ADDR: .BLKB 4

.PSECT \$CODE,NOWRT, PIC,2

		00FC	00000	PARSE_MAX	TABLE_SIZE:		
					.WORD	Save R2,R3,R4,R5,R6,R7	2133
57	00000000G	00	9E	00002	MOVAB	LIB\$SIGNAL, R7	
56	00000000G	00	9E	00009	MOVAB	GET_VM, R6	
55	000000000	00	9E	00010	MOVAB	DISK_DEVICES, R5	
54	000000000	00	9E	00017	MOVAB	SIZE, R4	
	F4	A4	D4	0001E	CLRL	CONTEXT	2177
		54	DD	00021	PUSHL	R4	2178
	08	A4	9F	00023	PUSHAB	VALUE_ADDR	
	FC	A4	9F	00026	PUSHAB	INDEX	
	F4	A4	9F	00029	PUSHAB	CONTEXT	
00000000V	00	04	FB	0002C	CALLS	#4, PARSE_TEXT_RECORD	
	01	50	E8	00033	BLBS	STATUS, 1\$	
			04	00036	RET		
		54	DD	00037	1\$: PUSHL	R4	2184
	08	A4	DD	00039	PUSHL	VALUE_ADDR	
		64	DD	0003C	PUSHL	SIZE	
00000000G	00	03	FB	0003E	CALLS	#3, LIB\$CVT_DTB	
	04	50	D0	00045	MOVL	R0, STATUS	
	10	A4	E8	00049	BLBS	STATUS, 2\$	2185
		04	A4	9F	0004D	PUSHAB	VALUE_ADDR
	08	64	DD	00050	PUSHL	SIZE	
		02	DD	00052	PUSHL	#2	
	00000000G	8F	DD	00054	PUSHL	#ERF_CVTERR	
67		04	FB	0005A	CALLS	#4, LIB\$SIGNAL	
		64	D6	0005D	2\$: INCL	SIZE	2190
	01	A4	D1	0005F	CMPL	INDEX, #1	2196
		4B	12	00063	BNEQ	3\$	
	50	64	D0	00065	MOVL	SIZE, R0	2199
3A	52	02	78	00068	ASHL	#2, R0, AMOUNT	
	A5	01	A3	0006C	SUBW3	#1, R0, MAX_CPU_TYPES	2200
		52	DD	00071	PUSHL	AMOUNT	2201
	66	01	FB	00073	CALLS	#1, GET_VM	
54	A5	50	D0	00076	MOVL	R0, MIN_RANGE_TABLE_ADDR	
		52	DD	0007A	PUSHL	AMOUNT	2202
	66	01	FB	0007C	CALLS	#1, GET_VM	
40	A5	50	D0	0007F	MOVL	R0, MAX_RANGE_TABLE_ADDR	
	52	02	C4	00083	MULL2	#2, R2	2203
		52	DD	00086	PUSHL	R2	
	66	01	FB	00088	CALLS	#1, GET_VM	
48	A5	50	D0	0008B	MOVL	R0, MIN_MODULES_DESC	
		52	DD	0008F	PUSHL	R2	2204
	66	01	FB	00091	CALLS	#1, GET_VM	
	4C	50	D0	00094	MOVL	R0, MAX_MODULES_DESC	
7E		64	01	78	00098	ASHL	#1, SIZE, -(SP) 2205

		66	01	FB	0009C	CALLS	#1, GET_VM		
		A5	50	D0	0009F	MOVL	R0, PROCESSOR_TYPE_TABLE		
7E	60	64	01	78	000A3	ASHL	#1, SIZE, -(SP)	2206	
		66	01	FB	000A7	CALLS	#1, GET_VM		
	50	A5	50	D0	000AA	MOVL	R0, MIN_MAX_TABLE_SIZES	2207	
			54	11	000AE	BRB	5\$	2214	
7E		64	01	78	000B0	ASHL	#1, SIZE, -(SP)		
		66	01	FB	000B4	CALLS	#1, GET_VM		
	F0	A4	50	D0	000B7	MOVL	R0, DEVICE_ADDR	2215	
7E		64	01	78	000BB	ASHL	#1, SIZE, -(SP)		
		66	01	FB	000BF	CALLS	#1, GET_VM		
	0C	A4	50	D0	000C2	MOVL	R0, VERSION_ADDR	2216	
7E		64	02	78	000C6	ASHL	#2, SIZE, -(SP)		
		66	01	FB	000CA	CALLS	#1, GET_VM		
	10	A4	50	D0	000CD	MOVL	R0, XFER_ADDR	2217	
7E		64	03	78	000D1	ASHL	#3, SIZE, -(SP)		
		66	01	FB	000D5	CALLS	#1, GET_VM		
	F8	A4	50	D0	000D8	MOVL	R0, IMAGE_ADDR	2224	
		50	FC	A4	D0	MOVL	INDEX, R0		
	07	02	50	CF	000E0	CASEL	R0, #2, #7		
00B7	0085	0053	0023		000E4	.WORD	6\$-4\$,-		
0195	0153	0120	00ED		000EC		7\$-4\$,-		
							8\$-4\$,-		
							10\$-4\$,-		
							12\$-4\$,-		
							14\$-4\$,-		
							16\$-4\$,-		
							18\$-4\$		
			58	A5	DD	000F4	PUSHL	MODULE_NAME_DESC	2307
			50	DD	000F7	PUSHL	R0		
			01	DD	000F9	PUSHL	#1		
		00000000G	8F	DD	000FB	PUSHL	#ERF_BADEVVAL		
	67		04	FB	00101	CALLS	#4, LIB\$SIGNAL		
			01DB	31	00104	BRW	22\$	2308	
	65		F0	A4	D0	00107	MOVL	DEVICE_ADDR, DISK_DEVICES	2226
	00000000G	00	0C	A4	D0	0010B	MOVL	VERSION_ADDR, DISK_VERSION	2227
	00000000G	00	10	A4	D0	00113	MOVL	XFER_ADDR, DISK_XFER_ADDR	2228
	00000000G	00	F8	A4	D0	0011B	MOVL	IMAGE_ADDR, DISK_IMAGE	2229
		50		64	D0	00123	MOVL	SIZE, R0	2230
3C	A5	50		01	83	00126	SUBB3	#1, R0, MAX_DISK_TYPE	
		51		65	D0	0012B	MOVL	DISK_DEVICES, R1	2231
		61		50	B0	0012E	MOVW	R0, (R1)	
	EC	A4		01	B0	00131	MOVW	#1, DC_CLASS	2232
				62	11	00135	BRB	9\$	2224
	7C	A5	F0	A4	D0	00137	MOVL	DEVICE_ADDR, TAPE_DEVICES	2236
	00000000G	00	0C	A4	D0	0013C	MOVL	VERSION_ADDR, TAPE_VERSION	2237
	00000000G	00	10	A4	D0	00144	MOVL	XFER_ADDR, TAPE_XFER_ADDR	2238
	00000000G	00	F8	A4	D0	0014C	MOVL	IMAGE_ADDR, TAPE_IMAGE	2239
		50		64	D0	00154	MOVL	SIZE, R0	2240
46	A5	50		01	83	00157	SUBB3	#1, R0, MAX_TAPE_TYPE	
		51	7C	A5	D0	0015C	MOVL	TAPE_DEVICES, R1	2241
		61		50	B0	00160	MOVW	R0, (R1)	
	EC	A4		02	B0	00163	MOVW	#2, DC_CLASS	2242
				66	11	00167	BRB	11\$	2224
	70	A5	F0	A4	D0	00169	MOVL	DEVICE_ADDR, SCOM_DEVICES	2246
	00000000G	00	0C	A4	D0	0016E	MOVL	VERSION_ADDR, SCOM_VERSION	2247
	00000000G	00	10	A4	D0	00176	MOVL	XFER_ADDR, SCOM_XFER_ADDR	2248

00000000G	00	F8	A4	D0	0017E	MOVL	IMAGE_ADDR, SCOM_IMAGE	2249
	50		64	D0	00186	MOVL	SIZE, R0	2250
45	A5		01	83	00189	SUBB3	#1, R0, MAX_SCOM_TYPE	
	51	70	A5	D0	0018E	MOVL	SCOM_DEVICES, R1	2251
	61		50	B0	00192	MOVW	R0, (R1)	
	EC		20	B0	00195	MOVW	#32, DC_CLASS	2252
	34		67	11	00199	BRB	13\$	2224
	A5	F0	A4	D0	0019B	MOVL	DEVICE_ADDR, LP_DEVICES	2256
00000000G	00	OC	A4	D0	001A0	MOVL	VERSION_ADDR, LP_VERSION	2257
00000000G	00	10	A4	D0	001A8	MOVL	XFER_ADDR, LP_XFER_ADDR	2258
00000000G	00	F8	A4	D0	001B0	MOVL	IMAGE_ADDR, LP_IMAGE	2259
	50		64	D0	001B8	MOVL	SIZE, R0	2260
00000000G	00		01	83	001BB	SUBB3	#1, R0, MAX_LP_TYPE	
	51	34	A5	D0	001C3	MOVL	LP_DEVICES, R1	2261
	61		50	B0	001C7	MOVW	R0, (R1)	
	EC		43	8F	9B	MOVZBW	#67, DC_CLASS	2262
	A4		64	11	001CF	BRB	15\$	2224
	64	F0	A4	D0	001D1	MOVL	DEVICE_ADDR, REALTIME_DEVICES	2266
00000000G	00	OC	A4	D0	001D6	MOVL	VERSION_ADDR, REALTIME_VERSION	2267
00000000G	00	10	A4	D0	001DE	MOVL	XFER_ADDR, REALTIME_XFER_ADDR	2268
00000000G	00	F8	A4	D0	001E6	MOVL	IMAGE_ADDR, REALTIME_IMAGE	2269
	50		64	D0	001EE	MOVL	SIZE, R0	2270
44	A5		01	83	001F1	SUBB3	#1, R0, MAX_REALTIME_TYPE	
	51	64	A5	D0	001F6	MOVL	REALTIME_DEVICES, R1	2271
	61		50	B0	001FA	MOVW	R0, (R1)	
	EC		60	8F	9B	MOVZBW	#96, DC_CLASS	2272
	A4		73	11	00202	BRB	17\$	2224
	F0	F0	A4	D0	00204	MOVL	DEVICE_ADDR, BUS_DEVICES	2276
00000000G	00	OC	A4	D0	00209	MOVL	VERSION_ADDR, BUS_VERSION	2277
00000000G	00	10	A4	D0	00211	MOVL	XFER_ADDR, BUS_XFER_ADDR	2278
00000000G	00	F8	A4	D0	00219	MOVL	IMAGE_ADDR, BUS_IMAGE	2279
	50		64	D0	00221	MOVL	SIZE, R0	2280
38	A5		01	83	00224	SUBB3	#1, R0, MAX_BUS_TYPE	
	51	F0	A5	D0	00229	MOVL	BUS_DEVICES, R1	2281
	61		50	B0	0022D	MOVW	R0, (R1)	
	EC		80	8F	9B	MOVZBW	#128, DC_CLASS	2282
	A4		75	11	00235	BRB	19\$	2224
	5C	F0	A4	D0	00237	MOVL	DEVICE_ADDR, PACKET_PROCESSOR_DEVICES	2286
00000000G	00	OC	A4	D0	0023C	MOVL	VERSION_ADDR, PACKET_PROCESSOR_VERSION	2287
00000000G	00	10	A4	D0	00244	MOVL	XFER_ADDR, PACKET_PROCESSOR_XFER_ADDR	2288
00000000G	00	F8	A4	D0	0024C	MOVL	IMAGE_ADDR, PACKET_PROCESSOR_IMAGE	2289
	51		64	D0	00254	MOVL	SIZE, R1	2290
00000000G	00		01	83	00257	SUBB3	#1, R1, MAX_MISC_TYPE	
	51	5C	A5	D0	0025F	MOVL	PACKET_PROCESSOR_DEVICES, R0	2291
	50		51	B0	00263	MOVW	R1, (R0)	
	60		01	78	00266	ASHL	#1, R1, -(SP)	2292
	7E		01	FB	0026A	CALLS	#1, GET_VM	
	66		50	D0	0026D	MOVL	R0, TRANSLATE_ENTRY_TABLE	
00000000G	00	EC	A4	B4	00274	CLRW	DC_CLASS	2293
			33	11	00277	BRB	19\$	2224
	00A4	F0	A4	D0	00279	MOVL	DEVICE_ADDR, WORKSTATION_DEVICES	2297
00000000G	00	OC	A4	D0	0027F	MOVL	VERSION_ADDR, WORKSTATION_VERSION	2298
00000000G	00	10	A4	D0	00287	MOVL	XFER_ADDR, WORKSTATION_XFER_ADDR	2299
00000000G	00	F8	A4	D0	0028F	MOVL	IMAGE_ADDR, WORKSTATION_IMAGE	2300
	50		64	D0	00297	MOVL	SIZE, R0	2301
47	A5		01	83	0029A	SUBB3	#1, R0, MAX_WORKSTATION_TYPE	
	51	00A4	C5	D0	0029F	MOVL	WORKSTATION_DEVICES, R1	2302

			61		50	B0	002A4		MOVW	R0, (R1)		
	EC		A4	46	8F	9B	002A7		MOVZBW	#70, DC CLASS	:	2303
			53	39	A5	9A	002AC	19\$:	MOVZBL	MAX_CLASSES, R3	:	2318
			52	EC	A4	3C	002B0		MOVZWL	DC CLASS, R2	:	2320
					50	D4	002B4		CLRL	COUNT	:	
					19	11	002B6		BRB	21\$:	
		51		FC	A5	D0	002B8	20\$:	MOVL	DEV CLASS PTR, R1	:	
					6140	3F	002BC		PUSHAW	(R1)[COUNT]	:	
52		10			00	ED	002BF		CMPZV	#0, #16, @ (SP)+, R2	:	
	9E				0B	12	002C4		BNEQ	21\$:	
		51		F8	A5	D0	002C6		MOVL	DEV ADDRS PTR, R1	:	2322
		6140		F0	A4	D0	002CA		MOVL	DEVICE_ADDR, (R1)[COUNT]	:	
					11	11	002CF		BRB	22\$:	2323
	E3	50			53	F3	002D1	21\$:	AOBLEQ	R3, COUNT, 20\$:	2318
					52	DD	002D5		PUSHL	R2	:	2327
					01	DD	002D7		PUSHL	#1	:	
			00000000G		8F	DD	002D9		PUSHL	#ERF CLSTBLERR	:	
		67			03	FB	002DF		CALLS	#3, [IB\$SIGNAL	:	
		50			01	D0	002E2	22\$:	MOVL	#1, R0	:	2328
					04	002E5			RET		:	2329

; Routine Size: 742 bytes, Routine Base: \$CODE + 0B6A

```
1782 2330 1 Routine PARSE_DEVICE_DESC_RECORD =
1783 2331 2 BEGIN
1784 2332 2 ++
1785 2333 2 Functional description :
1786 2334 2 Each call to 'PARSE_TEXT_RECORD' returns the next item in
1787 2335 2 the record (comma seperated list). The state of CONTEXT
1788 2336 2 determines if more items are available in the record. The
1789 2337 2 item count (I) does not consider the INDEX as a item in the
1790 2338 2 record. The first item in the record is the item after the
1791 2339 2 '='.
1792 2340 2 Record format:
1793 2341 2 INDEX = ITEM ONE,ITEM TWO,...ITEM FOUR
1794 2342 2 See 'DEVICES' module in SYS$LIBRARY:ERFLIB.TLB for
1795 2343 2 more information.
1796 2344 2
1797 2345 2 For the first and second items returned convert them to dec.
1798 2346 2
1799 2347 2 If this is ITEM ONE then save it as a device class. For
1800 2348 2 each device class (ITEM ONE) see if the device type (INDEX)
1801 2349 2 is greater the is max allowable value.
1802 2350 2
1803 2351 2 If this is ITEM TWO the save it as the allowable version
1804 2352 2 number for a loadable routine.
1805 2353 2
1806 2354 2 If this is ITEM THREE then it is a two character device
1807 2355 2 name. Use INDEX to obtain the address in which this string
1808 2356 2 should be copied to.
1809 2357 2
1810 2358 2 If this is ITEM FOUR then it is the name of the loadable image
1811 2359 2 that will interprt this deviced error packet. This string
1812 2360 2 is copied to a table indexed by INDEX.
1813 2361 2
1814 2362 2 Calling sequence
1815 2363 2
1816 2364 2 Input parameters
1817 2365 2
1818 2366 2 RECORD_DESC global descriptor pointing at the record read
1819 2367 2 from the text library.
1820 2368 2
1821 2369 2 Output parameters
1822 2370 2
1823 2371 2 None
1824 2372 2
1825 2373 2 Routine value
1826 2374 2
1827 2375 2 Worst error is returned.
1828 2376 2
1829 2377 2 ----
1830 2378 2
1831 2379 2 OWN
1832 2380 2 Context,
1833 2381 2 Dc_class: BYTE,
1834 2382 2 I,
1835 2383 2 Index,
1836 2384 2 Item_address: REF $BBLOCK[dsc$k_d_bln],
1837 2385 2 Size,
1838 2386 2 Status,
```


ERF
V04-000

Errorlog Report Formatter

F 7
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 67
(21)

: 1839	2387	2	Table_addr,
: 1840	2388	2	Temp,
: 1841	2389	2	Value_addr,
: 1842	2390	2	Version;
: 1843	2391	2	

```
1845 2392 2 Context = 0;      ! Clear the context
1846 2393 2 I = 1;           ! Set the item count
1847 2394 2
1848 2395 2 Do
1849 2396 2 Begin
1850 2397 3 CALL_FUNCTION ( Parse_text_record ( context, index, value_addr, size ) );
1851 2398 3
1852 2399 3 If .I LEQ 2 then      ! If first or second item
1853 2400 4 Begin              ! then convert to binary
1854 2401 4   Status = LIB$CVT_DTB ( .size, .Value_addr, size );
1855 2402 4   If NOT .status then Signal (erf_cvterr, 2, .size, value_addr) ;
1856 2403 4
1857 2404 4   If .I EQL 1 then dc_class = .size;  ! If first item save
1858 2405 3 End;
1859 2406 3
1860 2407 3 !
1861 2408 3 ! Determine which device class is being processed.
1862 2409 3 !
1863 2410 3 Selectoneu .dc_class of
1864 2411 3 Set
1865 2412 3
1866 2413 3 [DC$_ZERO_CLASS]:
1867 2414 4 Begin
1868 2415 4
1869 2416 4   ! Make sure it's a valid device type for this class of devices.
1870 2417 4
1871 2418 4   If .index GTR .max_misc_type then
1872 2419 4     Signal (erf_baddevtyp, 2, .index, .module_name_desc);
1873 2420 4
1874 2421 4   !
1875 2422 4   ! Determine which portion of the record is being processed and
1876 2423 4   ! get the address of the location to store the data.
1877 2424 4
1878 2425 4   Case .I from 2 to 4 of
1879 2426 4     Set
1880 2427 4     [2]:
1881 2428 4       Packet_processor_version [ .index ] = .size;
1882 2429 4     [3]:
1883 2430 4       Table_addr = Packet_processor_devices[.index];
1884 2431 4     [4]:
1885 2432 4       Table_addr = Packet_processor_image[.index, desc_one];
1886 2433 4   [OutOfRange]:
1887 2434 4   TES;
1888 2435 3 End ;
1889 2436 3
1890 2437 3 [DC$_DISK]:
1891 2438 4 Begin
1892 2439 4
1893 2440 4   ! Make sure it's a valid device type for this class of devices.
1894 2441 4
1895 2442 4   If .index GTR .max_disk_type then
1896 2443 4     Signal (erf_baddevtyp, 2, .index, .Module_name_desc);
1897 2444 4
1898 2445 4   !
1899 2446 4   ! Determine which portion of the record is being processed and
1900 2447 4   ! get the address of the location to store the data.
1901 2448 4
```



```

: 1902      2449  4      Case .I from 2 to 4 of
: 1903      2450  4      Set
: 1904      2451  4      [2]:
: 1905      2452  4      Disk_version [ .index ] = .size;
: 1906      2453  4      [3]:
: 1907      2454  4      Table_addr = Disk_devices[.index];
: 1908      2455  4      [4]:
: 1909      2456  4      Table_addr = Disk_image[.index, desc_one];
: 1910      2457  4      [Outrange]:
: 1911      2458  4      TES;
: 1912      2459  3      End ;
: 1913      2460  3
: 1914      2461  3      [DC$_TAPE]:
: 1915      2462  4      Begin
: 1916      2463  4      |
: 1917      2464  4      | Make sure it's a valid device type for this class of devices.
: 1918      2465  4      |
: 1919      2466  4      | If .index GTR .max_tape_type then
: 1920      2467  4      |   Signal (erf_badevtyp, 2, .index, .Module_name_desc);
: 1921      2468  4      |
: 1922      2469  4      | |
: 1923      2470  4      | | Determine which portion of the record is being processed and
: 1924      2471  4      | | get the address of the location to store the data.
: 1925      2472  4      | |
: 1926      2473  4      | | Case .I from 2 to 4 of
: 1927      2474  4      | | Set
: 1928      2475  4      | | [2]:
: 1929      2476  4      | |   Tape_version [ .index ] = .size;
: 1930      2477  4      | | [3]:
: 1931      2478  4      | |   Table_addr = tape_devices[.index];
: 1932      2479  4      | | [4]:
: 1933      2480  4      | |   Table_addr = Tape_image[.index, desc_one];
: 1934      2481  4      | | [Outrange]:
: 1935      2482  4      | | TES;
: 1936      2483  3      | | End ;
: 1937      2484  3      | |
: 1938      2485  3      | | [DC$_SCOM]:
: 1939      2486  4      | | Begin
: 1940      2487  4      | | |
: 1941      2488  4      | | | Make sure it's a valid device type for this class of devices.
: 1942      2489  4      | | |
: 1943      2490  4      | | | If .index GTR .max_scom_type
: 1944      2491  4      | | | Then
: 1945      2492  4      | | |   Signal (erf_badevtyp, 2, .index, .Module_name_desc) ;
: 1946      2493  4      | | |
: 1947      2494  4      | | | |
: 1948      2495  4      | | | | Determine which portion of the record is being processed and
: 1949      2496  4      | | | | get the address of the location to store the data.
: 1950      2497  4      | | | |
: 1951      2498  4      | | | | Case .I from 2 to 4 of
: 1952      2499  4      | | | | Set
: 1953      2500  4      | | | | [2]:
: 1954      2501  4      | | | |   Scom_version[.index] = .size ;
: 1955      2502  4      | | | |
: 1956      2503  4      | | | | [3]:
: 1957      2504  4      | | | |   Table_addr = scom_devices[.index];
: 1958      2505  4      | | | |
```

```
: 1959      2506  4      [4]:
: 1960      2507  4      Table_addr = scom_image[.index, desc_one];
: 1961      2508  4
: 1962      2509  4      [Outrange]:
: 1963      2510  4      TES ;
: 1964      2511  3      End ;
: 1965      2512  3
: 1966      2513  3      [DC$ LP]:
: 1967      2514  4      Begin
: 1968      2515  4      |
: 1969      2516  4      | Make sure it's a valid device type for this class of devices.
: 1970      2517  4
: 1971      2518  4      If .index GTR .max_lp_type
: 1972      2519  4      Then
: 1973      2520  4      Signal (erf_badevtyp, 2, .index, .Module_name_desc) ;
: 1974      2521  4
: 1975      2522  4      |
: 1976      2523  4      | Determine which portion of the record is being processed and
: 1977      2524  4      | get the address of the location to store the data.
: 1978      2525  4
: 1979      2526  4      Case .I from 2 to 4 of
: 1980      2527  4      Set
: 1981      2528  4      [2]:
: 1982      2529  4      Lp_version[.index] = .size ;
: 1983      2530  4
: 1984      2531  4      [3]:
: 1985      2532  4      Table_addr = lp_devices[.index];
: 1986      2533  4
: 1987      2534  4      [4]:
: 1988      2535  4      Table_addr = lp_image[.index, desc_one];
: 1989      2536  4
: 1990      2537  4      [Outrange]:
: 1991      2538  4      TES ;
: 1992      2539  3      End ;
: 1993      2540  3
: 1994      2541  3      [DC$ REALTIME]:
: 1995      2542  4      Begin
: 1996      2543  4      |
: 1997      2544  4      | Make sure it's a valid device type for this class of devices.
: 1998      2545  4
: 1999      2546  4      If .index GTR .max_realtime_type
: 2000      2547  4      Then
: 2001      2548  4      Signal (erf_badevtyp, 2, .index, .Module_name_desc) ;
: 2002      2549  4
: 2003      2550  4      |
: 2004      2551  4      | Determine which portion of the record is being processed and
: 2005      2552  4      | get the address of the location to store the data.
: 2006      2553  4
: 2007      2554  4      Case .I from 2 to 4 of
: 2008      2555  4      Set
: 2009      2556  4      [2]:
: 2010      2557  4      Realtime_version[.index] = .size ;
: 2011      2558  4
: 2012      2559  4      [3]:
: 2013      2560  4      Table_addr = realtime_devices[.index];
: 2014      2561  4
: 2015      2562  4      [4]:
```



```
: 2016      2563  4      Table_addr = realtime_image[.index, desc_one];
: 2017      2564  4
: 2018      2565  4      [Outrange]:
: 2019      2566  4      TES ;
: 2020      2567  3      End ;
: 2021      2568  3
: 2022      2569  3      [DC$_BUS]:
: 2023      2570  4      Begin
: 2024      2571  4
: 2025      2572  4      | Make sure it's a valid device type for this class of devices.
: 2026      2573  4
: 2027      2574  4      If .index GTR .max_bus_type
: 2028      2575  4      Then
: 2029      2576  4          Signal (erf_baddevtyp, 2, .index, .Module_name_desc) ;
: 2030      2577  4
: 2031      2578  4
: 2032      2579  4      | Determine which portion of the record is being processed and
: 2033      2580  4      | get the address of the location to store the data.
: 2034      2581  4
: 2035      2582  4      Case .I from 2 to 4 of
: 2036      2583  4      Set
: 2037      2584  4      [2]:
: 2038      2585  4          Bus_version[.index] = .size ;
: 2039      2586  4
: 2040      2587  4      [3]:
: 2041      2588  4          Table_addr = bus_devices[.index];
: 2042      2589  4
: 2043      2590  4      [4]:
: 2044      2591  4          Table_addr = bus_image[.index, desc_one];
: 2045      2592  4
: 2046      2593  4      [Outrange]:
: 2047      2594  4      TES ;
: 2048      2595  3      End ;
: 2049      2596  3
: 2050      2597  3      [DC$_WORKSTATION]:
: 2051      2598  4      Begin
: 2052      2599  4
: 2053      2600  4      | Make sure it's a valid device type for this class of devices.
: 2054      2601  4
: 2055      2602  4      If .index GTR .max_workstation_type then
: 2056      2603  4          Signal (erf_baddevtyp, 2, .index, .Module_name_desc);
: 2057      2604  4
: 2058      2605  4
: 2059      2606  4      | Determine which portion of the record is being processed and
: 2060      2607  4      | get the address of the location to store the data.
: 2061      2608  4
: 2062      2609  4      Case .I from 2 to 4 of
: 2063      2610  4      Set
: 2064      2611  4      [2]:
: 2065      2612  4          Workstation_version [ .index ] = .size;
: 2066      2613  4      [3]:
: 2067      2614  4          Table_addr = Workstation_devices[.index];
: 2068      2615  4      [4]:
: 2069      2616  4          Table_addr = Workstation_image[.index, desc_one];
: 2070      2617  4      [Outrange]:
: 2071      2618  4      TES;
: 2072      2619  3      End ;
```

```
2073 2620 3
2074 2621 3 TES;
2075 2622 3
2076 2623 3 If .I EQL 3 then CH$MOVE (.size, .value_addr, .table_addr );
2077 2624 3
2078 2625 3 If .I EQL 4 then
2079 2626 4 Begin
2080 2627 4 Item_address = .table_addr;
2081 2628 4 Item_address[dsc$w_length] = .size ;
2082 2629 4 Item_address[dsc$a_pointer] = get_vm(.size);
2083 2630 4 CH$MOVE (.size, .value_addr, .item_address[dsc$a_pointer]);
2084 2631 4 End;
2085 2632 3
2086 2633 3 I = .I + 1;
2087 2634 3
2088 2635 3 End
2089 2636 3
2090 2637 2 While .context EQL 1;
2091 2638 2
2092 2639 2 Return true ;
2093 2640 1 End ;
```

.PSECT \$OWNS,NOEXE, PIC,2

```
00058 CONTEXT:.BLKB 4
0005C DC_CLASS:
0005D .BLKB 1
00060 I: .BLKB 3
00064 INDEX: .BLKB 4
00068 ITEM_ADDRESS:
00069 .BLKB 4
0006C SIZE: .BLKB 4
00070 STATUS: .BLKB 4
00074 TABLE_ADDR:
00075 .BLKB 4
00078 TEMP: .BLKB 4
0007C VALUE_ADDR:
0007D .BLKB 4
00080 VERSION:.BLKB 4
```

.PSECT \$CODE,NOWRT, PIC,2

```
07FC 00000 PARSE_DEVICE_DESC_RECORD:
5A 00000000G 8F D0 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10 : 2330
59 00000000G 00 9E 00009 MOVL #ERF_BADEVTYP, R10
58 00000000' 00 9E 00010 MOVAB LIB$SIGNAL, R9
57 00000000' 00 9E 00017 MOVAB MODULE_NAME_DESC, R8
F8 A7 D4 0001E CLRL CONTEXT : 2392
67 01 D0 00021 MOVL #1, I : 2393
OC A7 9F 00024 1$: PUSHAB SIZE : 2397
1C A7 9F 00027 PUSHAB VALUE_ADDR
```


00000000V	00	04	A7	9F	0002A	PUSHAB	INDEX	:	
	01	F8	A7	9F	0002D	PUSHAB	CONTEXT	:	
	02		04	FB	00030	CALLS	#4, PARSE_TEXT_RECORD	:	
			50	E8	00037	BLBS	STATUS, 2\$:	
				04	0003A	RET		:	
			67	D1	0003B	2\$:	CMPL	I, #2	2399
			33	14	0003E	BGTR	4\$:	
		0C	A7	9F	00040	PUSHAB	SIZE	:	2401
		1C	A7	DD	00043	PUSHL	VALUE_ADDR	:	
		0C	A7	DD	00046	PUSHL	SIZE	:	
00000000G	00		03	FB	00049	CALLS	#3, LIB\$CVT_DTB	:	
10	A7		50	D0	00050	MOVL	R0, STATUS	:	
	11	10	A7	E8	00054	BLBS	STATUS, 3\$:	2402
		1C	A7	9F	00058	PUSHAB	VALUE_ADDR	:	
		0C	A7	DD	0005B	PUSHL	SIZE	:	
			02	DD	0005E	PUSHL	#2	:	
		00000000G	8F	DD	00060	PUSHL	#ERF_CVTERR	:	
	69		04	FB	00066	CALLS	#4, LIB\$SIGNAL	:	
	01		67	D1	00069	3\$:	CMPL	I, #1	2404
			05	12	0006C	BNEQ	4\$:	
FC	A7	0C	A7	90	0006E	MOVB	SIZE, DC_CLASS	:	
	50	FC	A7	9A	00073	4\$:	MOVZBL	DC_CLASS, R0	2410
			3E	12	00077	BNEQ	10\$:	2413
	51	04	A7	D0	00079	MOVL	INDEX, R1	:	2418
51 00000000G 00	08		00	ED	0007D	CMPZV	#0, #8, MAX_MISC_TYPE, R1	:	
			0B	18	00086	BGEQ	5\$:	
			68	DD	00088	PUSHL	MODULE_NAME_DESC	:	2419
			51	DD	0008A	PUSHL	R1	:	
			02	DD	0008C	PUSHL	#2	:	
			5A	DD	0008E	PUSHL	R10	:	
	69		04	FB	00090	CALLS	#4, LIB\$SIGNAL	:	
	02		67	CF	00093	5\$:	CASEL	I, #2, #2	2425
0017	0011	0008			00097	6\$:	.WORD	7\$-6\$,-	
								8\$-6\$,-	
								9\$-6\$	
			7E	11	0009D	BRB	19\$:	
	51	00000000G	00	D0	0009F	7\$:	MOVL	PACKET_PROCESSOR_VERSION, R1	2428
			7E	11	000A6	BRB	21\$:	
	51	04	A8	D0	000A8	8\$:	MOVL	PACKET_PROCESSOR_DEVICES, R1	2430
			7E	11	000AC	BRB	23\$:	
	51	00000000G	00	D0	000AE	9\$:	MOVL	PACKET_PROCESSOR_IMAGE, R1	2432
			7E	11	000B5	BRB	25\$:	
	01		50	91	000B7	10\$:	CMPB	R0, #1	2437
			3B	12	000BA	BNEQ	16\$:	
	51	04	A7	D0	000BC	MOVL	INDEX, R1	:	2442
51 E4 A8	08		00	ED	000C0	CMPZV	#0, #8, MAX_DISK_TYPE, R1	:	
			0B	18	000C6	BGEQ	11\$:	
			68	DD	000C8	PUSHL	MODULE_NAME_DESC	:	2443
			51	DD	000CA	PUSHL	R1	:	
			02	DD	000CC	PUSHL	#2	:	
			5A	DD	000CE	PUSHL	R10	:	
	69		04	FB	000D0	CALLS	#4, LIB\$SIGNAL	:	
	02		67	CF	000D3	11\$:	CASEL	I, #2, #2	2449
0017	0011	0008			000D7	12\$:	.WORD	13\$-12\$,-	
								14\$-12\$,-	
								15\$-12\$	
			7E	11	000DD	BRB	29\$:	

51		00000000G	00	DO	000DF	13\$:	MOVL	DISK_VERSION, R1	2452
			7E	11	000E6		BRB	31\$	
51		A8	A8	DO	000E8	14\$:	MOVL	DISK_DEVICES, R1	2454
			7E	11	000EC		BRB	33\$	
51		00000000G	00	DO	000EE	15\$:	MOVL	DISK_IMAGE, R1	2456
			7E	11	000F5		BRB	35\$	
02			50	91	000F7	16\$:	CMPB	R0, #2	2461
			3B	12	000FA		BNEQ	26\$	
51		04	A7	DO	000FC		MOVL	INDEX, R1	2466
08			00	ED	00100		CMPZV	#0, #8, MAX_TAPE_TYPE, R1	
			0B	18	00106		BGEQ	17\$	
			68	DD	00108		PUSHL	MODULE_NAME_DESC	2467
			51	DD	0010A		PUSHL	R1	
			02	DD	0010C		PUSHL	#2	
			5A	DD	0010E		PUSHL	R10	
69			04	FB	00110		CALLS	#4, LIB\$SIGNAL	
02			67	CF	00113	17\$:	CASEL	1, #2, #2	2473
0017		0011	0008		00117	18\$:	.WORD	20\$-18\$,- 22\$-18\$,- 24\$-18\$	
			3E	11	0011D	19\$:	BRB	29\$	
51		00000000G	00	DO	0011F	20\$:	MOVL	TAPE_VERSION, R1	2476
			3E	11	00126	21\$:	BRB	31\$	
51		24	A8	DO	00128	22\$:	MOVL	TAPE_DEVICES, R1	2478
			3E	11	0012C	23\$:	BRB	33\$	
51		00000000G	00	DO	0012E	24\$:	MOVL	TAPE_IMAGE, R1	2480
			3E	11	00135	25\$:	BRB	35\$	
20			50	91	00137	26\$:	CMPB	R0, #32	2485
			3B	12	0013A		BNEQ	36\$	
51		04	A7	DO	0013C		MOVL	INDEX, R1	2490
08			00	ED	00140		CMPZV	#0, #8, MAX_SCOM_TYPE, R1	
			0B	18	00146		BGEQ	27\$	
			68	DD	00148		PUSHL	MODULE_NAME_DESC	2492
			51	DD	0014A		PUSHL	R1	
			02	DD	0014C		PUSHL	#2	
			5A	DD	0014E		PUSHL	R10	
69			04	FB	00150		CALLS	#4, LIB\$SIGNAL	
02			67	CF	00153	27\$:	CASEL	1, #2, #2	2498
0017		0011	0008		00157	28\$:	.WORD	30\$-28\$,- 32\$-28\$,- 34\$-28\$	
			42	11	0015D	29\$:	BRB	39\$	
51		00000000G	00	DO	0015F	30\$:	MOVL	SCOM_VERSION, R1	2501
			42	11	00166	31\$:	BRB	41\$	
51		18	A8	DO	00168	32\$:	MOVL	SCOM_DEVICES, R1	2504
			42	11	0016C	33\$:	BRB	43\$	
51		00000000G	00	DO	0016E	34\$:	MOVL	SCOM_IMAGE, R1	2507
			42	11	00175	35\$:	BRB	45\$	
43		8F	50	91	00177	36\$:	CMPB	R0, #67	2513
			3E	12	0017B		BNEQ	46\$	
51		04	A7	DO	0017D		MOVL	INDEX, R1	2518
08			00	ED	00181		CMPZV	#0, #8, MAX_LP_TYPE, R1	
			0B	18	0018A		BGEQ	37\$	
			68	DD	0018C		PUSHL	MODULE_NAME_DESC	2520
			51	DD	0018E		PUSHL	R1	
			02	DD	00190		PUSHL	#2	
			5A	DD	00192		PUSHL	R10	

			69	04	FB	00194		CALLS	#4, LIB\$SIGNAL		
	02		02	67	CF	00197	37\$:	CASEL	1, #2, #2		2526
	0017		0011	0008		0019B	38\$:	.WORD	40\$-38\$,-		
									42\$-38\$,-		
									44\$-38\$		
									49\$		
			51	00000000G	3F	11	001A1	39\$:	BRB		
					00	D0	001A3	40\$:	MOVL	LP VERSION, R1	2529
					3F	11	001AA	41\$:	BRB	51\$	
			51	DC	A8	D0	001AC	42\$:	MOVL	LP DEVICES, R1	2532
					3F	11	001B0	43\$:	BRB	53\$	
			51	00000000G	00	D0	001B2	44\$:	MOVL	LP IMAGE, R1	2535
					3F	11	001B9	45\$:	BRB	55\$	
			60	8F	50	91	001BB	46\$:	CMPB	R0, #96	2541
					3B	12	001BF		BNEQ	56\$	
			51		A7	D0	001C1		MOVL	INDEX, R1	2546
51	EC	A8	08		00	ED	001C5		CMPZV	#0, #8, MAX_REALTIME_TYPE, R1	
					0B	18	001CB		BGEQ	47\$	
					68	DD	001CD		PUSHL	MODULE_NAME_DESC	2548
					51	DD	001CF		PUSHL	R1	
					02	DD	001D1		PUSHL	#2	
					5A	DD	001D3		PUSHL	R10	
			69		04	FB	001D5		CALLS	#4, LIB\$SIGNAL	
	02		02		67	CF	001D8	47\$:	CASEL	1, #2, #2	2554
	0017		0011		0008		001DC	48\$:	.WORD	50\$-48\$,-	
										52\$-48\$,-	
										54\$-48\$	
										59\$	
			51	00000000G	3F	11	001E2	49\$:	BRB		
					00	D0	001E4	50\$:	MOVL	REALTIME_VERSION, R1	2557
					3F	11	001EB	51\$:	BRB	61\$	
			51	0C	A8	D0	001ED	52\$:	MOVL	REALTIME_DEVICES, R1	2560
					3F	11	001F1	53\$:	BRB	63\$	
			51	00000000G	00	D0	001F3	54\$:	MOVL	REALTIME_IMAGE, R1	2563
					3F	11	001FA	55\$:	BRB	65\$	
			80	8F	50	91	001FC	56\$:	CMPB	R0, #128	2569
					3B	12	00200		BNEQ	66\$	
			50		A7	D0	00202		MOVL	INDEX, R0	2574
50	E0	A8	08		00	ED	00206		CMPZV	#0, #8, MAX_BUS_TYPE, R0	
					0B	18	0020C		BGEQ	57\$	
					68	DD	0020E		PUSHL	MODULE_NAME_DESC	2576
					50	DD	00210		PUSHL	R0	
					02	DD	00212		PUSHL	#2	
					5A	DD	00214		PUSHL	R10	
			69		04	FB	00216		CALLS	#4, LIB\$SIGNAL	
	02		02		67	CF	00219	57\$:	CASEL	1, #2, #2	2582
	0017		0011		0008		0021D	58\$:	.WORD	60\$-58\$,-	
										62\$-58\$,-	
										64\$-58\$	
										75\$	
			51	00000000G	72	11	00223	59\$:	BRB		
					00	D0	00225	60\$:	MOVL	BUS_VERSION, R1	2585
					3F	11	0022C	61\$:	BRB	70\$	
			51	98	A8	D0	0022E	62\$:	MOVL	BUS_DEVICES, R1	2588
					48	11	00232	63\$:	BRB	72\$	
			51	00000000G	00	D0	00234	64\$:	MOVL	BUS_IMAGE, R1	2591
					51	11	0023B	65\$:	BRB	74\$	
			46	8F	50	91	0023D	66\$:	CMPB	R0, #70	2597
					54	12	00241		BNEQ	75\$	
			50		A7	D0	00243		MOVL	INDEX, R0	2602

50	EF	A8	08	00	ED	00247	CMPZV	#0, #8, MAX_WORKSTATION_TYPE, R0			
				08	18	0024D	BGEQ	67\$			
				68	DD	0024F	PUSHL	MODULE_NAME_DESC	2603		
				50	DD	00251	PUSHL	R0			
				02	DD	00253	PUSHL	#2			
				5A	DD	00255	PUSHL	R10			
		69		04	FB	00257	CALLS	#4, LIB\$SIGNAL			
	02	02		67	CF	0025A	CASEL	1, #2, #2	2609		
	0029	001A		0008		0025E	.WORD	69\$-68\$,- 71\$-68\$,- 73\$-68\$			
				31	11	00264	BRB	75\$			
		51	00000000G	00	D0	00266	MOVL	WORKSTATION_VERSION, R1	2612		
		50	04	A7	D0	0026D	MOVL	INDEX, R0			
		6140	0C	A7	B0	00271	MOVW	SIZE, (R1)[R0]			
				1F	11	00276	BRB	75\$			
		51	4C	A8	D0	00278	MOVL	WORKSTATION_DEVICES, R1	2614		
		50	04	A7	D0	0027C	MOVL	INDEX, R0			
	14	A7		6140	3E	00280	MOVAW	(R1)[R0], TABLE_ADDR			
				10	11	00285	BRB	75\$			
		51	00000000G	00	D0	00287	MOVL	WORKSTATION_IMAGE, R1	2616		
		50	04	A7	D0	0028E	MOVL	INDEX, R0			
		14	A7	6140	7E	00292	MOVAW	(R1)[R0], TABLE_ADDR			
		56		67	D0	00297	MOVL	I, R6	2623		
		03		56	D1	0029A	CMPL	R6, #3			
				0D	12	0029D	BNEQ	76\$			
		51	1C	A7	D0	0029F	MOVL	VALUE_ADDR, R1			
		50	14	A7	D0	002A3	MOVL	TABLE_ADDR, R0			
	60	61	0C	A7	28	002A7	MOVW	SIZE, (R1), (R0)			
		04		56	D1	002AC	CMPL	R6, #4	2625		
				2B	12	002AF	BNEQ	77\$			
		08	A7	14	A7	D0	002B1	MOVL	TABLE_ADDR, ITEM_ADDRESS	2627	
		52	08	A7	D0	002B6	MOVL	ITEM_ADDRESS, R2	2628		
		50	0C	A7	D0	002BA	MOVL	SIZE, R0			
		62		50	B0	002BE	MOVW	R0, (R2)			
				50	DD	002C1	PUSHL	R0	2629		
		00000000G	00	01	FB	002C3	CALLS	#1, GET_VM			
		04	A2	50	D0	002CA	MOVL	R0, 4(R2)			
			51	1C	A7	D0	002CE	MOVL	VALUE_ADDR, R1	2630	
			50	08	A7	D0	002D2	MOVL	ITEM_ADDRESS, R0		
	04	B0		61	0C	A7	28	002D6	MOVW	SIZE, (R1), @4(R0)	
				67	D6	002DC	INCL	I	2633		
			01	F8	A7	D1	002DE	CMPL	CONTEXT, #1	2637	
				03	12	002E2	BNEQ	78\$			
				FD3D	31	002E4	BRW	1\$			
			50	01	D0	002E7	MOVL	#1, R0	2639		
				04	002EA	RET			2640		

; Routine Size: 747 bytes, Routine Base: \$CODE + 0E50


```
2095 2641 1 Routine PARSE_MODULE_NAMES =
2096 2642 2 Begin
2097 2643 3 ++
2098 2644 4 Functional description
2099 2645 5
2100 2646 6     This routine builds a descriptor table, which contains the names
2101 2647 7     of library modules to be processed.
2102 2648 8
2103 2649 9 Calling sequence
2104 2650 10
2105 2651 11 Input parameters
2106 2652 12
2107 2653 13     None.
2108 2654 14
2109 2655 15 Output parameters
2110 2656 16
2111 2657 17 Routine value
2112 2658 18
2113 2659 19 -----
2114 2660 20
2115 2661 21 Local
2116 2662 22
2117 2663 23     Desc: REF $bblock [],
2118 2664 24     Context: initial (0),
2119 2665 25     Index,
2120 2666 26     Value_addr,
2121 2667 27     Size;
2122 2668 28
2123 2669 29 Do
2124 2670 30 Begin
2125 2671 31
2126 2672 32     Item_count = .item_count + 1;
2127 2673 33
2128 2674 34     If .item_count GTR .table_length then
2129 2675 35         (signal (erf_badevtyp, 2, .item_count, .Module_name_desc); Return true);
2130 2676 36
2131 2677 37     Call_function (Parse_text_record ( context, index, value_addr, size) );
2132 2678 38
2133 2679 39     Desc = desc_table_address[.item_count, desc_one];
2134 2680 40
2135 2681 41     Desc[dsc$w_length] = .size;
2136 2682 42     Desc[dsc$b_class] = dsc$k_class_d;
2137 2683 43     Desc[dsc$b_dtype] = dsc$k_dtype_t;
2138 2684 44     ! SIZE could be zero if the lib. module had ".," or "<EOL>"
2139 2685 45     ! This is could be a problem.
2140 2686 46     Desc[dsc$a_pointer] = get_vm(.size);
2141 2687 47
2142 2688 48     CH$MOVE (.desc[dsc$w_length], .value_addr, .desc[dsc$a_pointer]);
2143 2689 49
2144 2690 50 End
2145 2691 51 While .context EQL 1;
2146 2692 52
2147 2693 53 Return true;
2148 2694 54
2149 2695 55 End;
```

```
00FC 00000 PARSE_MODULE NAMES:
57 00000000' 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7 : 2641
5E          10 C2 00009 MOVAB ITEM_COUNT, R7 :
          0C AE D4 0000C SUBL2 #16, SP : 2642
          67 D6 0000F 1$: INCL CONTEXT : 2673
50          67 D0 00011 MOVL ITEM_COUNT, R0 : 2675
08          00 ED 00014 CMPZV #0, #8, TABLE_LENGTH, R0 :
          16 18 0001A BGEQ 2$ :
          34 A7 DD 0001C PUSHL MODULE_NAME_DESC : 2676
          50 DD 0001F PUSHL R0 :
          02 DD 00021 PUSHL #2 :
          8F DD 00023 PUSHL #ERF_BADEVTYP :
          04 FB 00029 CALLS #4, [IB$SIGNAL :
          42 11 00030 BRB 3$ :
          5E DD 00032 2$: PUSHL SP : 2678
          08 AE 9F 00034 PUSHAB VALUE_ADDR :
          10 AE 9F 00037 PUSHAB INDEX :
          18 AE 9F 0003A PUSHAB CONTEXT :
          04 FB 0003D CALLS #4, PARSE_TEXT_RECORD :
          50 E9 00044 BLBC STATUS, 4$ :
          51 D0 A7 D0 00047 MOVL DESC_TABLE_ADDRESS, R1 : 2680
          50 67 D0 0004B MOVL ITEM_COUNT, R0 :
          56 6140 7E 0004E MOVAQ (R1)[R0], DESC :
          66 6E B0 00052 MOVW SIZE, (DESC) : 2682
          02 A6 020E 8F B0 00055 MOVW #526, 2(DESC) : 2684
          6E DD 0005B PUSHL SIZE : 2687
          00 01 FB 0005D CALLS #1, GET_VM :
          04 A6 50 D0 00064 MOVL R0, 4(DESC) :
          04 BE 66 28 00068 MOV3 (DESC), @VALUE_ADDR, @4(DESC) : 2689
          01 0C AE D1 0006E CMPL CONTEXT, #1 : 2692
          9B 13 00072 BEQL 1$ :
          50 01 D0 00074 3$: MOVL #1, R0 : 2694
          04 00077 4$: RET : 2695
```

; Routine Size: 120 bytes, Routine Base: \$CODE + 113B

; 2150 2696 1


```
2152 2697 1 Routine PARSE_TEXT_RECORD ( context, index, value_addr, size ) =
2153 2698 2 BEGIN
2154 2699 2 ++
2155 2700 2 Functional description
2156 2701 2
2157 2702 2 This routine parses a record that was previously read from
2158 2703 2 the text library. Each call to this routine returns the next
2159 2704 2 item in the record(comma seperated list). CONTEXT is set after
2160 2705 2 returning all items in the list. The value of INDEX, in binary,
2161 2706 2 is constant for all items in a record. VALUE_ADDR is the starting
2162 2707 2 address of the next item. Its size is returned in SIZE. All records
2163 2708 2 processed by this routine are expected to have 4 items after there
2164 2709 2 index. See 'DEVICES' module in SYS$LIBRARY:ERFLIB.TLB for more
2165 2710 2 information.
2166 2711 2
2167 2712 2
2168 2713 2 Calling sequence
2169 2714 2
2170 2715 2 Input parameters
2171 2716 2
2172 2717 2 Context : Should always be zero in first call to this routine
2173 2718 2 on return from this routine it is set to one to
2174 2719 2 specify that there more values in the list.
2175 2720 2
2176 2721 2
2177 2722 2 Output parameters
2178 2723 2
2179 2724 2 Index : Binary value of the number to the left of the equal
2180 2725 2 sign.
2181 2726 2
2182 2727 2 Value_addr : Starting address of the string to be returned.
2183 2728 2
2184 2729 2 Size : The length of the field pointed to by VALUE_ADDR
2185 2730 2
2186 2731 2 Context : Binary 1 to indicate more values in the comma
2187 2732 2 separated list.
2188 2733 2 Binary 0 to indicate no more in the list.
2189 2734 2
2190 2735 2
2191 2736 2 Routine value
2192 2737 2
2193 2738 2 Worst error is returned.
2194 2739 2
2195 2740 2 ----
2196 2741 2
2197 2742 2 LITERAL
2198 2743 2 Max_deliminters = 3,
2199 2744 2 TAB = 9;
2200 2745 2
2201 2746 2 OWN
2202 2747 2 Context_length,
2203 2748 2 Context_pointer,
2204 2749 2 Delim_position: INITIAL (0),
2205 2750 2 Length_to_move,
2206 2751 2 Offset,
2207 2752 2 Status,
2208 2753 2 Temp_ptr;
```

ERF
V04-000

Errorlog Report Formatter

F 8
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 80
(24)

: 2209
: 2210

2754 2
2755 2


```
2212 2756 2 |
2213 2757 2 | If context equals 1 then compress the record and get the index.
2214 2758 2 |
2215 2759 2 |
2216 2760 2 | If ..context EQL 0 then
2217 2761 2 | Begin
2218 2762 2 |
2219 2763 2 |
2220 2764 2 |     Setup pointer to start of record and record length.
2221 2765 2 |
2222 2766 2 |     Context_pointer = CH$PTR ( .Record_desc [dsc$a_pointer] );
2223 2767 2 |     Context_length = .Record_desc [dsc$w_length];
2224 2768 2 |
2225 2769 2 |
2226 2770 2 |     Search the record for a '!'.
2227 2771 2 |
2228 2772 2 |     Delim_position = CH$FIND_CH ( .context_length, .context_pointer, %c'!' );
2229 2773 2 |
2230 2774 2 |
2231 2775 2 |     If the '!' is in the first character position then this is a comment,
2232 2776 2 |     so return to get another record. If it is not in the first position
2233 2777 2 |     then reset the record length to the '!' position.
2234 2778 2 |
2235 2779 2 | If .Delim_position NEQ 0 then
2236 2780 2 | Begin
2237 2781 2 |     Offset = CH$DIFF ( .delim_position, .context_pointer );
2238 2782 2 |     If .Offset LEQ 3 then
2239 2783 2 |         Return false
2240 2784 2 |     else
2241 2785 2 |         Context_length = .Offset - 1 ;
2242 2786 2 |     End;
2243 2787 2 |
2244 2788 2 | Offset = 0;
2245 2789 2 |
2246 2790 2 |
2247 2791 2 |     Search the record for a ' '. If a blank then compress record.
2248 2792 2 |
2249 2793 2 |     Delim_position = 1;
2250 2794 2 |     While .Delim_position NEQ 0 do
2251 2795 2 |         Begin
2252 2796 2 |             Delim_position = CH$FIND_CH ( .context_length, .context_pointer, %C' ');
2253 2797 2 |             If .Delim_position EQL 0 then
2254 2798 2 |                 Delim_position = CH$FIND_CH ( .context_length, .context_pointer, TAB );
2255 2799 2 |             If .Delim_position NEQ 0 then
2256 2800 2 |                 Begin
2257 2801 2 |                     offset = .offset + 1;
2258 2802 2 |                     Temp_ptr = CH$DIFF ( .delim_position + 1, .context_pointer );
2259 2803 2 |                     Length_to_move = .context_length - .temp_ptr ;
2260 2804 2 |                     Temp_ptr = CH$COPY ( .length_to_move, .delim_position + 1, %C'*.
2261 2805 2 |                                     .length_to_move + 1, .delim_position );
2262 2806 2 |                 End;
2263 2807 2 |             End;
2264 2808 2 |
2265 2809 2 | Context_length = .context_length - .offset;
2266 2810 2 |
```

```

00084 CONTEXT_LENGTH:
      .BLKB 4
00088 CONTEXT_POINTER:
      .BLKB 4
00000000 0008C DELIM_POSITION:
      .LONG 0
00090 LENGTH_TO_MOVE:

```


00094 OFFSET: .BLKB 4
00098 STATUS: .BLKB 4
0009C TEMP_PTR: .BLKB 4

.PSECT \$CODE,NOWRT, PIC,2

07FC 00000 PARSE_TEXT RECORD:

	5A	00000000'	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10	: 2697	
	59	04	AC	D0	00009	MOVAB	DELIM_POSITION, R10	: 2760	
			69	D5	0000D	MOVL	CONTEXT, R9		
			03	13	0000F	TSTL	(R9)		
			00E1	31	00011	BEQL	1\$		
						BRW	12\$		
FC	AA	00000000'	00	D0	00014	1\$: MOVL	RECORD_DESC+4, CONTEXT_POINTER	: 2766	
F8	AA	00000000'	00	3C	0001C	MOVZWL	RECORD_DESC, CONTEXT_LENGTH	: 2767	
	57	FC	AA	D0	00024	MOVL	CONTEXT_POINTER, R7	: 2772	
67	F8	AA	21	3A	00028	LOCC	#33, CONTEXT_LENGTH, (R7)		
			02	12	0002D	BNEQ	2\$		
			51	D4	0002F	CLRL	R1		
	6A		51	D0	00031	2\$: MOVL	R1, DELIM_POSITION		
	50		6A	D0	00034	MOVL	DELIM_POSITION, R0	: 2779	
			16	13	00037	BEQL	4\$		
08	AA		50	C3	00039	SUBL3	R7, R0, OFFSET	: 2781	
			50	AA	0003E	MOVL	OFFSET, R0	: 2782	
			03	D1	00042	CMPL	R0, #3		
			03	14	00045	BGTR	3\$		
			0100	31	00047	BRW	18\$		
	F8	AA	FF	A0	9E	0004A	3\$: MOVAB	-1(R0), CONTEXT_LENGTH	: 2785
			08	AA	D4	0004F	4\$: CLRL	OFFSET	: 2788
	6A		01	D0	00052	MOVL	#1, DELIM_POSITION	: 2793	
	56		6A	D0	00055	MOVL	DELIM_POSITION, R6	: 2794	
	58		F8	AA	D0	00058	MOVL	CONTEXT_LENGTH, R8	: 2796
			56	D5	0005C	5\$: TSTL	R6	: 2794	
			47	13	0005E	BEQL	9\$		
67		58	20	3A	00060	LOCC	#32, R8, (R7)	: 2796	
			02	12	00064	BNEQ	6\$		
			51	D4	00066	CLRL	R1		
	6A		51	D0	00068	6\$: MOVL	R1, DELIM_POSITION		
			08	12	0006B	BNEQ	8\$: 2797	
67		58	09	3A	0006D	LOCC	#9, R8, (R7)	: 2798	
			02	12	00071	BNEQ	7\$		
			51	D4	00073	CLRL	R1		
	6A		51	D0	00075	7\$: MOVL	R1, DELIM_POSITION		
	56		6A	D0	00078	8\$: MOVL	DELIM_POSITION, R6	: 2799	
			DF	13	0007B	BEQL	5\$		
			08	AA	D6	0007D	INCL	OFFSET	: 2801
	50		57	C3	00080	SUBL3	R7, R6, R0	: 2802	
		10	56	AA	9E	00084	MOVAB	1(R0), TEMP_PTR	
04	AA		58	AA	C3	00089	SUBL3	TEMP_PTR, R8, LENGTH_TO_MOVE	: 2803
			51	AA	D0	0008F	MOVL	LENGTH_TO_MOVE, R1	: 2804
			50	56	D0	00093	MOVL	R6, R0	
			52	A1	9E	00096	MOVAB	1(R1), R2	: 2805
52	2A	01	A0	51	2C	0009A	MOVCS	R1, 1(R0), #42, R2, (R6)	

; Routine Size: 333 bytes, Routine Base: \$CODE + 11B3


```
2314 2856 1 Routine INIT_COMMONS =
2315 2857 2 Begin
2316 2858 2
2317 2859 2 ++
2318 2860 2
2319 2861 2 Functional Description:
2320 2862 2
2321 2863 2 This routine initializes some of the commons in
2322 2864 2 ERFSHR (qiocommon, opcodes, modes).
2323 2865 2
2324 2866 2 Calling Sequence:
2325 2867 2
2326 2868 2 Init_commons ()
2327 2869 2
2328 2870 2 Input parameters
2329 2871 2
2330 2872 2 None
2331 2873 2
2332 2874 2 Output parameters
2333 2875 2
2334 2876 2 None
2335 2877 2
2336 2878 2 --
2337 2879 2
2338 2880 2 LOCAL
2339 2881 2 Array_addr,
2340 2882 2 Array_size,
2341 2883 2 Status,
2342 2884 2 Xfer_addr ;
2343 2885 2
2344 2886 2
2345 2887 2
2346 2888 2 Get the image name and attempt to load it.
2347 2889 2 Determine if a loading error occurred and signal it
2348 2890 2 if necessary.
2349 2891 2
2350 2892 2 Status = Map_image ( AD ('SYS$SYSTEM:ERFINICOM.EXE'), xfer_addr) ;
2351 2893 2 If NOT .status then return false ;
2352 2894 2
2353 2895 2
2354 2896 2 Execute the image. Then set the flag indicateing that the commons have been
2355 2897 2 initialized.
2356 2898 2
2357 2899 2 EXEC_IMAGE (xfer_addr) ;
2358 2900 2
2359 2901 2 Initied_commons = true ;
2360 2902 2
2361 2903 2 Return true ;
2362 2904 1 End ; ! Routine
```

.PSECT \$PLIT,NOWRT,NOEXE, PIC,2

```
49 46 52 45 3A 4D 45 54 53 59 53 24 53 59 53 001D4 P.ABV: .ASCII \SYS$SYSTEM:ERFINICOM.EXE\
45 58 45 2E 4D 4F 43 49 4E 001E3
00000018 001EC P.ABU: .LONG 24
```

00000000' 001F0

.ADDRESS P.ABV

;

.PSECT \$CODE,NOWRT, PIC,2

		0000 00000	INIT_COMMONS:			
	5E		04 C2 00002	.WORD	Save nothing	; 2856
			5E DD 00005	SUBL2	#4, SP	; 2892
		00000000'	00 9F 00007	PUSHL	SP	; 2893
00000000G	00		02 FB 0000D	PUSHAB	P.ABU	; 2899
	14		50 E9 00014	CALLS	#2, MAP_IMAGE	; 2901
			5E DD 00017	BLBC	STATUS,-1\$; 2903
00000000G	00		01 FB 00019	PUSHL	SP	; 2904
00000000'	00		01 D0 00020	CALLS	#1, EXEC_IMAGE	
	50		01 D0 00027	MOVL	#1, INITED_COMMONS	
			04 0002A	MOVL	#1, R0	
			50 D4 0002B 1\$:	RET	R0	
			04 0002D	CLRL		
				RET		

; Routine Size: 46 bytes, Routine Base: \$CODE + 1300

; 2363 2905 1


```
: 2365      2906 1 Global routine VALIDATE_PACKET =
: 2366      2907 2 BEGIN
: 2367      2908 2 ++
: 2368      2909 2 Functional description
: 2369      2910 2
: 2370      2911 2           This routine checks the error log packet entry type
: 2371      2912 2           to see if the type is valid for the CPU type it was
: 2372      2913 2           logged on.
: 2373      2914 2
: 2374      2915 2 Calling sequence
: 2375      2916 2
: 2376      2917 2     Validate_packet()
: 2377      2918 2
: 2378      2919 2 Input parameters
: 2379      2920 2
: 2380      2921 2
: 2381      2922 2 Output parameters
: 2382      2923 2
: 2383      2924 2     If valid_cpu, valid_class, valid_type or valid_entry are false
: 2384      2925 2     then return false.
: 2385      2926 2
: 2386      2927 2 Routine value
: 2387      2928 2
: 2388      2929 2     Worst error is returned.
: 2389      2930 2
: 2390      2931 2 ----
: 2391      2932 2
: 2392      2933 2 ! THERE SHOULD BE A MAX_ENTRY_TYPE.
: 2393      2934 2
: 2394      2935 2 GLOBAL
: 2395      2936 2     Processor_type:      LONG,
: 2396      2937 2     Device_class:        BYTE,
: 2397      2938 2     Device_type:         BYTE ;
: 2398      2939 2
: 2399      2940 2 LOCAL
: 2400      2941 2     Table_size:          WORD INITIAL (0),
: 2401      2942 2     Max_value:            BYTE,
: 2402      2943 2     Min_range:            REF VECTOR[WORD],
: 2403      2944 2     Max_range:            REF VECTOR[WORD],
: 2404      2945 2     Begin_bit_pos:        LONG INITIAL (24),
: 2405      2946 2     Version:              REF VECTOR[WORD],
: 2406      2947 2     Field_size:          LONG INITIAL (8);
```

```
2408 2948 2 |
2409 2949 2 | Set default state for valid flags
2410 2950 2 |
2411 2951 2 | Syecom[sye$b_Valid_CPU] = true;
2412 2952 2 | Syecom[sye$b_Valid_class] = true;
2413 2953 2 |
2414 2954 2 |
2415 2955 2 | Obtain processor type. Determine if the processor type in the SID has
2416 2956 2 | been set up. (Early VAX systems did not have the processor type set).
2417 2957 2 |
2418 2958 2 | Processor_type = LIB$EXTZV ( Begin_bit_pos, Field_size, emb[emb$l_hd_sid] );
2419 2959 2 |
2420 2960 2 | If .processor_type EQLU 255 then processor_type = 1;
2421 2961 2 |
2422 2962 2 |
2423 2963 2 |
2424 2964 2 | Depending on processor type, determine which set of tables to use.
2425 2965 2 | These tables specify invalid entry type ranges for specific cpu's.
2426 2966 2 |
2427 2967 2 | Incr loop_count from 1 to .max_cpu_types do
2428 2968 2 |   If .processor_type_table[.loop_count] EQL .Processor_type then
2429 2969 2 |     Begin
2430 2970 2 |       Min_range = .Min_range_table_addr[.loop_count];
2431 2971 2 |       Max_range = .Max_range_table_addr[.loop_count];
2432 2972 2 |       Table_size = .Min_max_table_sizes[.loop_count];
2433 2973 2 |       Exitloop;
2434 2974 2 |     End;
2435 2975 2 |
2436 2976 2 | If .table_size EQL 0 then syecom[sye$b_Valid_CPU] = False;
```



```
2438 2977 2 |
2439 2978 2 | Ensure processing a device type entry.
2440 2979 2 |
2441 2980 2 | If DEVICE_TYPE_ENTRY ( )
2442 2981 2 | Then
2443 2982 2 |   Begin
2444 2983 2 |     Determine the type of device entry and set up device class
2445 2984 2 |     and type from the appropriate fields in the EMB buffer.
2446 2985 2 |
2447 2986 2 |     Selectoneu .emb[emb$w_hd_entry] of
2448 2987 2 |       Set
2449 2988 2 |       [EMB$C_DE, EMB$C_DT, EMB$C_DA]:
2450 2989 2 |         Begin
2451 2990 2 |         Device_class = .emb[emb$b_dv_class] ;
2452 2991 2 |         Device_type = .emb[emb$b_dv_type] ;
2453 2992 2 |         End ;
2454 2993 2 |
2455 2994 2 |       [EMB$C_LM]:
2456 2995 2 |       Begin
2457 2996 2 |       Device_class = .emb[emb$b_lm_class] ;
2458 2997 2 |       Device_type = .emb[emb$b_lm_type] ;
2459 2998 2 |       End ;
2460 2999 2 |
2461 3000 2 |       [EMB$C_SP]:
2462 3001 2 |       Begin
2463 3002 2 |       Device_class = .emb[emb$b_sp_class] ;
2464 3003 2 |       Device_type = .emb[emb$b_sp_type] ;
2465 3004 2 |       End ;
2466 3005 2 |
2467 3006 2 |       [EMB$K_LOGMSCP]:
2468 3007 2 |       Begin
2469 3008 2 |       If CH$EQL (2,emb[driver_type],2,CH$PTR(uplit('DISK'))))
2470 3009 2 |       Then
2471 3010 2 |       Begin
2472 3011 2 |       Device_class = DC$_DISK ;
2473 3012 2 |       Device_type = 1 ;
2474 3013 2 |       End ;
2475 3014 2 |
2476 3015 2 |       If CH$EQL (2,emb[driver_type],2,CH$PTR(uplit('TAPE'))))
2477 3016 2 |       Then
2478 3017 2 |       Begin
2479 3018 2 |       Device_class = DC$_TAPE ;
2480 3019 2 |       Device_type = 1 ;
2481 3020 2 |       End ;
2482 3021 2 |       End ;
2483 3022 2 |
2484 3023 2 |     Tes ;
2485 3024 2 |
2486 3025 2 |     Determine the device class and set up the maximum number
2487 3026 2 |     of device types.
2488 3027 2 |     If class is out of range then VALID_class = False
2489 3028 2 |
2490 3029 2 |     Selectoneu .device_class of
2491 3030 2 |     Set
2492 3031 2 |
2493 3032 2 |
2494 3033 2 |
```

```

: 2495      3034 3      [DC$_DISK]:      ! Disk
: 2496      3035 4      BEGIN
: 2497      3036 4      Max_value = .max_disk_type;
: 2498      3037 4      Version = .disk_version;
: 2499      3038 4      END;
: 2500      3039 3
: 2501      3040 3      [DC$_TAPE]:      ! Tape
: 2502      3041 4      BEGIN
: 2503      3042 4      Max_value = .max_tape_type;
: 2504      3043 4      Version = .tape_version;
: 2505      3044 4      END;
: 2506      3045 3
: 2507      3046 3      [DC$_SCOM]:      ! Scom
: 2508      3047 4      BEGIN
: 2509      3048 4      Max_value = .max_scom_type;
: 2510      3049 4      Version = .scom_version;
: 2511      3050 4      END;
: 2512      3051 3
: 2513      3052 3      [DC$_LP]:      ! Printers
: 2514      3053 4      BEGIN
: 2515      3054 4      Max_value = .max_lp_type;
: 2516      3055 4      Version = .lp_version;
: 2517      3056 4      END;
: 2518      3057 3
: 2519      3058 3      [DC$_REALTIME]: ! Realtime
: 2520      3059 4      BEGIN
: 2521      3060 4      Max_value = .max_realtime_type;
: 2522      3061 4      Version = .realtime_version;
: 2523      3062 4      END;
: 2524      3063 3
: 2525      3064 3      [DC$_BUS]:      ! Buses
: 2526      3065 4      BEGIN
: 2527      3066 4      Max_value = .max_bus_type;
: 2528      3067 4      Version = .bus_version;
: 2529      3068 4      END;
: 2530      3069 3
: 2531      3070 3      [DC$_WORKSTATION]: ! Workstations
: 2532      3071 4      BEGIN
: 2533      3072 4      Max_value = .max_workstation_type;
: 2534      3073 4      Version = .workstation_version;
: 2535      3074 4      END;
: 2536      3075 3
: 2537      3076 3      [OTHERWISE]:
: 2538      3077 4      Begin
: 2539      3078 4      Max_value = 0;
: 2540      3079 4      Version = 0;
: 2541      3080 4      Syecom[syeb_Valid_class] = false;
: 2542      3081 4      End;
: 2543      3082 3
: 2544      3083 3      TES;
: 2545      3084 3
: 2546      3085 3
: 2547      3086 3      If device type is less than 1 or greater than max
: 2548      3087 3      value or the version number is zero, then set flags false.
: 2549      3088 3
: 2550      3089 3      If ( .device_type LSSU 1 ) OR
: 2551      3090 3      ( .device_type GTRU .max_value ) OR
```



```

: 2552      3091      4      ( .version EQLU 0 )
: 2553      3092      4      Then
: 2554      3093      4      Syecom[sye$b_Valid_type] = false
: 2555      3094      4      Else
: 2556      3095      4      If .version[.device_type] EQLU 0
: 2557      3096      4      then
: 2558      3097      4      Syecom[sye$b_Valid_type] = false
: 2559      3098      4      else
: 2560      3099      4      Syecom[sye$b_Valid_type] = true;
: 2561      3100      4      End
: 2562      3101      4      Else
: 2563      3102      4      Syecom[sye$b_valid_type] = true ;
: 2564      3103      4
: 2565      3104      4
: 2566      3105      4      :
: 2567      3106      4      Ensure a valid cpu type was found. Otherwise don't attempt
: 2568      3107      4      to do entry type verification.
: 2569      3108      4      :
: 2570      3109      4      If .syecom[sye$b_valid_cpu]
: 2571      3110      4      Then
: 2572      3111      4      Begin
: 2573      3112      4      Incr I from 1 to .table_size do
: 2574      3113      4      Begin
: 2575      3114      4      If ( .emb[emb$w_hd_entry] GEQU .min_range[I] ) AND
: 2576      3115      4      ( .emb[emb$w_hd_entry] LEQU .max_range[I] )
: 2577      3116      4      then
: 2578      3117      4      Begin
: 2579      3118      4      Syecom[sye$b_Valid_entry] = false ;
: 2580      3119      4      Exitloop;
: 2581      3120      4      End
: 2582      3121      4      else
: 2583      3122      4      Syecom[sye$b_Valid_entry] = true ;
: 2584      3123      4      End;
: 2585      3124      4      End;
: 2586      3125      4      End ;
: 2587      3126      4
: 2588      3127      4      If NOT .syecom[sye$b_valid_cpu] OR
: 2589      3128      4      NOT .syecom[sye$b_valid_class] OR
: 2590      3129      4      NOT .syecom[sye$b_valid_type] OR
: 2591      3130      4      NOT .syecom[sye$b_valid_entry]
: 2592      3131      4      Then return false;
: 2593      3132      4
: 2594      3133      4      Return true;
: 2595      3134      4      End;

```

.PSECT \$PLIT,NOWRT,NOEXE, PIC,2

```

4B 53 49 44 001F4 P.ABW: .ASCII \DISK\
45 50 41 54 001F8 P.ABX: .ASCII \TAPE\

```

.PSECT \$GLOBAL\$,NOEXE, PIC,2

```

000C0 PROCESSOR_TYPE::
      .BLKB 4
000C4 DEVICE_CLASS::

```

.BLKB 1
000C5 DEVICE_TYPE::
.BLKB 1

.PSECT \$CODE,NOWRT, PIC,2

			01FC 00000		.ENTRY	VALIDATE PACKET, Save R2,R3,R4,R5,R6,R7,R8	: 2906
	58	00000000G	00 9E 00002		MOVAB	EMB+4, R8	
	57	00000000G	00 9E 00009		MOVAB	SYECOM+25, R7	
	56	00000000'	00 9E 00010		MOVAB	DEVICE_CLASS, R6	
			52 B4 00017		CLRW	TABLE_SIZE	: 2907
			18 DD 00019		PUSHL	#24	
			08 DD 0001B		PUSHL	#8	
	67	0101	8F B0 0001D		MOVW	#257, SYECOM+25	: 2952
		FC	A8 9F 00022		PUSHAB	EMB	: 2958
		04	AE 9F 00025		PUSHAB	FIELD_SIZE	
		0C	AE 9F 00028		PUSHAB	BEGIN_BIT_POS	
	00000000G	00	03 FB 0002B		CALLS	#3, LIB\$EXTZV	
	FC	A6	50 D0 00032		MOVL	R0, PROCESSOR_TYPE	
	000000FF	8F	A6 D1 00036		CMPL	PROCESSOR_TYPE, #255	: 2960
			04 12 0003E		BNEQ	1\$	
	FC	A6	01 D0 00040		MOVL	#1, PROCESSOR_TYPE	
		53	A6 3C 00044	1\$:	MOVZWL	MAX_CPU_TYPES, R3	: 2967
			50 D4 00048		CLRL	LOOP_COUNT	: 2968
			29 11 0004A		BRB	3\$	
		51	AC A6 D0 0004C	2\$:	MOVL	PROCESSOR_TYPE_TABLE, R1	
			6140 3F 00050		PUSHAW	(R1)[LOOP_COUNT]	
FC	A6	9E	00 ED 00053		CMPZV	#0, #16, 8(SP)+, PROCESSOR_TYPE	
			1A 12 00059		BNEQ	3\$	
		51	A0 A6 D0 0005B		MOVL	MIN_RANGE_TABLE_ADDR, R1	: 2970
		55	6140 D0 0005F		MOVL	(R1)[LOOP_COUNT], MIN_RANGE	
		51	8C A6 D0 00063		MOVL	MAX_RANGE_TABLE_ADDR, R1	: 2971
		54	6140 D0 00067		MOVL	(R1)[LOOP_COUNT], MAX_RANGE	
		51	9C A6 D0 0006B		MOVL	MIN_MAX_TABLE_SIZES, R1	: 2972
		52	6140 B0 0006F		MOVW	(R1)[LOOP_COUNT], TABLE_SIZE	
			04 11 00073		BRB	4\$: 2969
		50	53 F3 00075	3\$:	AOBLEQ	R3, LOOP_COUNT, 2\$: 2968
D3		53	52 3C 00079	4\$:	MOVZWL	TABLE_SIZE, R3	: 2976
			03 12 0007C		BNEQ	5\$	
			01 A7 94 0007E		CLRB	SYECOM+26	
	00000000G	00	00 FB 00081	5\$:	CALLS	#0, DEVICE_TYPE_ENTRY	: 2980
		03	50 EB 00088		BLBS	R0, 6\$	
			00FE 31 0008B		BRW	22\$	
		50	68 3C 0008E	6\$:	MOVZWL	EMB+4, R0	: 2987
		01	50 B1 00091		CMPW	R0, #1	: 2989
			0E 13 00094		BEQL	7\$	
	0060	8F	50 B1 00096		CMPW	R0, #96	
			07 13 0009B		BEQL	7\$	
	0062	8F	50 B1 0009D		CMPW	R0, #98	
			06 12 000A2		BNEQ	8\$	
		66	18 A8 B0 000A4	7\$:	MOVW	EMB+28, DEVICE_CLASS	: 2991
			3B 11 000A8		BRB	12\$: 2987
	0064	8F	50 B1 000AA	8\$:	CMPW	R0, #100	: 2995
			07 13 000AF		BEQL	9\$	
	0063	8F	50 B1 000B1		CMPW	R0, #99	: 3001

		06	12	000B6	BNEQ	10\$		
	66	OC	A8	B0 000B8	9\$: MOVW	EMB+16, DEVICE_CLASS		3003
			27	11 000BC	BRB	12\$		2987
0065	8F		50	B1 000BE	10\$: CMPW	R0, #101		3007
			20	12 000C3	BNEQ	12\$		
	50	OE	A8	3C 000C5	MOVZWL	EMB+18, R0		3009
	50	00000000	00	B1 000C9	CMPW	P.ABW, R0		
			05	12 000D0	BNEQ	11\$		
	66	0101	8F	B0 000D2	MOVW	#257, DEVICE_CLASS		3012
	50	00000000	00	B1 000D7	11\$: CMPW	P.ABX, R0		3016
			05	12 000DE	BNEQ	12\$		
	66	0102	8F	B0 000E0	MOVW	#258, DEVICE_CLASS		3019
	50		66	9A 000E5	12\$: MOVZBL	DEVICE_CLASS, R0		3031
	01		50	91 000E8	CMPB	R0, #1		3034
			0D	12 000EB	BNEQ	13\$		
	52	88	A6	90 000ED	MOVB	MAX DISK TYPE, MAX VALUE		3036
	51	00000000G	00	D0 000F1	MOVL	DISK_VERSION, VERSION		3037
			79	11 000F8	BRB	20\$		3031
	02		50	91 000FA	13\$: CMPB	R0, #2		3040
			0D	12 000FD	BNEQ	14\$		
	52	92	A6	90 000FF	MOVB	MAX TAPE TYPE, MAX VALUE		3042
	51	00000000G	00	D0 00103	MOVL	TAPE_VERSION, VERSION		3043
			67	11 0010A	BRB	20\$		3031
	20		50	91 0010C	14\$: CMPB	R0, #32		3046
			0D	12 0010F	BNEQ	15\$		
	52	91	A6	90 00111	MOVB	MAX SCOM TYPE, MAX VALUE		3048
	51	00000000G	00	D0 00115	MOVL	SCOM_VERSION, VERSION		3049
			55	11 0011C	BRB	20\$		3031
43	8F		50	91 0011E	15\$: CMPB	R0, #67		3052
			10	12 00122	BNEQ	16\$		
	52	00000000G	00	90 00124	MOVB	MAX LP TYPE, MAX VALUE		3054
	51	00000000G	00	D0 0012B	MOVL	LP_VERSION, VERSION		3055
			3F	11 00132	BRB	20\$		3031
60	8F		50	91 00134	16\$: CMPB	R0, #96		3058
			0D	12 00138	BNEQ	17\$		
	52	90	A6	90 0013A	MOVB	MAX REALTIME_TYPE, MAX VALUE		3060
	51	00000000G	00	D0 0013E	MOVL	REALTIME_VERSION, VERSION		3061
			2C	11 00145	BRB	20\$		3031
80	8F		50	91 00147	17\$: CMPB	R0, #128		3064
			0D	12 0014B	BNEQ	18\$		
	52	84	A6	90 0014D	MOVB	MAX BUS TYPE, MAX VALUE		3066
	51	00000000G	00	D0 00151	MOVL	BUS_VERSION, VERSION		3067
			19	11 00158	BRB	20\$		3031
46	8F		50	91 0015A	18\$: CMPB	R0, #70		3070
			0D	12 0015E	BNEQ	19\$		
	52	93	A6	90 00160	MOVB	MAX WORKSTATION TYPE, MAX VALUE		3072
	51	00000000G	00	D0 00164	MOVL	WORKSTATION_VERSION, VERSION		3073
			06	11 0016B	BRB	20\$		3031
			52	94 0016D	19\$: CLRB	MAX VALUE		3078
			51	D4 0016F	CLRL	VERSION		3079
			67	94 00171	CLRB	SYECOM+25		3080
	50	01	A6	9A 00173	20\$: MOVZBL	DEVICE_TYPE, R0		3089
			0E	13 00177	BEQL	21\$		
	50		52	91 00179	CMPB	MAX_VALUE, R0		3090
			09	1F 0017C	BLSSU	21\$		
			51	D5 0017E	TSTL	VERSION		3091
			05	13 00180	BEQL	21\$		

51	9E	10	51	03	A7	37	51	01	03	6140	B5	00182	TSTW	(VERSION)[R0]	:	3095	
										05	12	00185	BNEQ	22\$:		
										03	A7	94	00187	21\$: CLRB	SYECOM+28	:	3097
										04	11	0018A	BRB	23\$:		
										01	01	90	0018C	22\$: MOVB	#1, SYECOM+28	:	3102
										01	A7	E9	00190	23\$: BLBC	SYECOM+26, 28\$:	3109
										68	3C	00194	MOVZWL	EMB+4, R1	:	3115	
										50	D4	00197	CLRL	I	:		
										19	11	00199	BRB	26\$:		
										6540	3F	0019B	24\$: PUSHAW	(MIN_RANGE)[I]	:		
										00	ED	0019E	CMPZV	#0, #16, a(SP)+, R1	:		
										0B	1A	001A3	BGTRU	25\$:		
										6440	B1	001A5	CMPW	(MAX_RANGE)[I], R1	:	3116	
										05	1F	001A9	BLSSU	25\$:		
										02	A7	94	001AB	CLRB	SYECOM+27	:	3119
										08	11	001AE	BRB	27\$:	3118	
										01	90	001B0	25\$: MOVB	#1, SYECOM+27	:	3123	
										53	F3	001B4	26\$: AOBLEQ	R3, I, 24\$:	3112	
										01	A7	E9	001B8	27\$: BLBC	SYECOM+26, 28\$:	3127
										67	E9	001BC	BLBC	SYECOM+25, 28\$:	3128	
										03	A7	E9	001BF	BLBC	SYECOM+28, 28\$:	3129
										02	A7	E9	001C3	BLBC	SYECOM+27, 28\$:	3130
										01	D0	001C7	MOVL	#1, R0	:	3133	
										04	04	001CA	RET		:		
										50	D4	001CB	28\$: CLRL	R0	:	3134	
										04	04	001CD	RET		:		

; Routine Size: 462 bytes, Routine Base: \$CODE + 132E


```
2597 3135 1 Routine HANDLER (sig, mech) =
2598 3136 1
2599 3137 1 ---
2600 3138 1
2601 3139 1 This condition handler gets control on any signalled
2602 3140 1 condition in order to save the highest severity error
2603 3141 1 to be returned by exit from the image.
2604 3142 1
2605 3143 1 Inputs:
2606 3144 1
2607 3145 1 signal_args = Address of signal argument list
2608 3146 1 mechanism_args = Address of mechanism argument list
2609 3147 1
2610 3148 1 Outputs:
2611 3149 1
2612 3150 1 WORST_ERROR is updated with highest severity error.
2613 3151 1
2614 3152 1 ---
2615 3153 1
2616 3154 2 BEGIN
2617 3155 2
2618 3156 2 External worst_error: $BBLOCK [LONG] ; ! Holds worst error encountered
2619 3157 2
2620 3158 2 MAP
2621 3159 2 sig: REF $BBLOCK, ! Standard VMS condition handler parameters.
2622 3160 2 mech: REF $BBLOCK; ! Address of signal argument list
2623 3161 2 ! Address of mechanism argument list
2624 3162 2
2625 3163 2 BIND
2626 3164 2 COND = SIG[CHF$L_SIG_NAME]: $BBLOCK ;! Condition
2627 3165 2
2628 3166 2
2629 3167 2 If .COND eql RMS$_EOF then return true;
2630 3168 2
2631 3169 2 If .cond[sts$v_fac_no] eql erf$_facility then
2632 3170 2 return ss$_resignal;
2633 3171 2
2634 3172 2 If
2635 3173 2 .cond[sts$v_severity] gtru .worst_error [sts$v_severity]
2636 3174 2 then
2637 3175 2 worst_error = .cond or sts$m_inhib_msg;
2638 3176 2
2639 3177 2 sig[chf$l_sig_args] = .sig[chf$l_sig_args] - 2; ! Dont count pc/psl
2640 3178 2 $putmsg (msgvec = sig[chf$l_sig_args], actrtn = write_err_msg);
2641 3179 2 sig[chf$l_sig_args] = .sig[chf$l_sig_args] + 2;
2642 3180 2
2643 3181 2 ss$_resignal ! Continue signalling
2644 3182 2
2645 3183 1 END;
```

```
53 00000000G 00 000C 00000 HANDLER: .WORD Save R2,R3
52 04 AC DO 00002 MOVAB WORST_ERROR, R3
MOVL SIG, R2
```

```
: 3135
: 3163
```

ERF
V04-000

Errorlog Report Formatter

I 9
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 96
(31)

			0001827A	8F	04	A2	D1	0000D		CMPL	4(R2), #98938	:	3167
				50		04	12	00015		BNEQ	1\$:	
						01	D0	00017		MOVL	#1, R0	:	
							04	0001A		RET		:	
08	06	A2		0C		00	ED	0001B	1\$:	CMPZV	#0, #12, 6(R2), #8	:	3169
						2D	13	00021		BEQL	3\$:	
50		63		03		00	EF	00023		EXTZV	#0, #3, WORST_ERROR, R0	:	3173
50	04	A2		03		00	ED	00028		CMPZV	#0, #3, 4(R2), R0	:	
						09	1B	0002E		BLEQU	2\$:	
		63	04	A2	10000000	8F	C9	00030		BISL3	#268435456, 4(R2), WORST_ERROR	:	3175
				62		02	C2	00039	2\$:	SUBL2	#2, (R2)	:	3177
						7E	7C	0003C		CLRQ	-(SP)	:	3178
					00000000V	00	9F	0003E		PUSHAB	WRITE_ERR_MSG	:	
						52	DD	00044		PUSHL	R2	:	
		00000000G		00		04	FB	00046		CALLS	#4, SYSSPUTMSG	:	
				62		02	C0	0004D		ADDL2	#2, (R2)	:	3179
				50	0918	8F	3C	00050	3\$:	MOVZWL	#2328, R0	:	3183
						04	00055			RET		:	

; Routine Size: 86 bytes, Routine Base: \$CODE + 14FC


```
2647 3184 1 Routine WRITE_ERR_MSG (Error_msg_desc) =
2648 3185 1 ---
2649 3186 1
2650 3187 1 This routine writes the error message to the output file.
2651 3188 1
2652 3189 1 Inputs:
2653 3190 1
2654 3191 1 error_msg_desc = Address of descriptor for message
2655 3192 1
2656 3193 1 Outputs:
2657 3194 1
2658 3195 1
2659 3196 1 ---
2660 3197 2 Begin
2661 3198 2
2662 3199 2 Local
2663 3200 2 Rmerror;
2664 3201 2
2665 3202 2 Map
2666 3203 2 Error_msg_desc : REF BLOCK[,BYTE];
2667 3204 2
2668 3205 2 If .Lstlun_rab_address EQL 0 then return false;
2669 3206 2
2670 3207 2 Lstlun_rab_address[rab$l_rbf] = .error_msg_desc[dsc$a_pointer];
2671 3208 2 Lstlun_rab_address[rab$w_rsz] = .error_msg_desc[dsc$w_length];
2672 3209 2
2673 3210 2 If NOT (rmerror = $put(rab = .Lstlun_rab_address)) then
2674 3211 2 ( Signal (.rmerror); Return .rmerror);
2675 3212 2
2676 3213 2 Return false;
2677 3214 1 End;
```

```
                                .EXTRN  SYS$PUT
                                0004 00000 WRITE_ERR MSG:
                                .WORD
                                Save R2
                                LSTLUN_RAB_ADDRESS, R1
                                3184
                                3205
                                51 00000000G 00 D0 00002 MOVL
                                BEQL 1$
                                3207
                                28 50 04 AC D0 0000B MOVL ERROR_MSG_DESC, R0
                                22 A1 04 A0 D0 0000F MOVL 4(R0), 40(R1)
                                3208
                                00000000G 00 01 FB 0001A PUSHL (R0), 34(R1)
                                52 50 D0 00021 CALLS R1
                                0D 52 E8 00024 #1, SYS$PUT
                                3210
                                00000000G 00 52 DD 00027 MOVL R0, RMSError
                                01 FB 00029 BLBS RMSError, 1$
                                52 D0 00030 PUSHL RMSError
                                3211
                                04 00033 CALLS #1, LIB$SIGNAL
                                50 D4 00034 1$: MOVL RMSError, R0
                                04 00036 RET
                                3214
```

; Routine Size: 55 bytes, Routine Base: \$CODE + 1552

```
2679 3215 1 Global routine WRITE_BINARY (BUFFER, RAB) =
2680 3216 1
2681 3217 1 ----
2682 3218 1
2683 3219 1 Functional description
2684 3220 1
2685 3221 1 This routine accepts a pointer to a buffer and writes
2686 3222 1 the buffer to an output stream in binary format.
2687 3223 1
2688 3224 1 Input parameters
2689 3225 1
2690 3226 1 BUFFER = address of an input record buffer
2691 3227 1
2692 3228 1 RAB = address of output rab
2693 3229 1
2694 3230 1 ----
2695 3231 1
2696 3232 2 BEGIN
2697 3233 2
2698 3234 2 MAP
2699 3235 2 rab: ref $bblock, ! Pointer to rab
2700 3236 2 buffer: ref $bblock; ! Describe the input buffer
2701 3237 2
2702 3238 2 LOCAL
2703 3239 2 desc: vector [2, long]; ! Temporary string descriptor
2704 3240 2
2705 3241 2
2706 3242 2 If .rab eql 0 then return true; ! Exit immediately if no output
2707 3243 2
2708 3244 2
2709 3245 2 ! INITIALIZE THE RAB
2710 3246 2 Store the buffer address and length in the RAB.
2711 3247 2
2712 3248 2
2713 3249 2 rab [rab$l_rbf] = .buffer; ! Store buffer address in RAB
2714 3250 2 rab [rab$w_rsz] = .input_rab[rab$w_rsz]; ! Store buffer size in RAB
2715 3251 2
2716 3252 2
2717 3253 2
2718 3254 2
2719 3255 2 ! WRITE TO FILE ---
2720 3256 2 Output the buffer via RMS.
2721 3257 2
2722 3258 2
2723 P 3259 2 CALL_FUNCTION ($put ( ! Call RMS with
2724 P 3260 2 rab = .rab, ! -record stream identifier
2725 3261 2 err = log_filename)); ! -error action routine
2726 3262 2
2727 3263 2 return true;
2728 3264 1 END;
```

5E

0000 00000
08 C2 00002.ENTRY WRITE_BINARY, Save nothing
SUBL2 #8, SP: 3215
:

ERF
V04-000

Errorlog Report Formatter

L 9
15-Sep-1984 23:42:14
14-Sep-1984 12:27:17

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[ERF.SRC]ERF.B32;1

Page 99
(33)

50	08	AC	DO	00005	MOVL	RAB, R0	: 3242
		1F	13	00009	BEQL	1\$: 3249
28	A0	04	AC	DO	0000B	MOVL	BUFFER, 40(R0)
22	A0	00000000G	00	B0	00010	MOVW	INPUT_RAB+34, 34(R0)
		00000000G	00	9F	00018	PUSHAB	LOG_FILENAME
			50	DD	0001E	PUSHL	R0
00000000G	00		02	FB	00020	CALLS	#2, SYSS\$PUT
	03		50	E9	00027	BLBC	STATUS, 2\$
	50		01	DO	0002A	MOVL	#1, R0
			04	0002D	2\$: RET		: 3263
							: 3264

; Routine Size: 46 bytes, Routine Base: \$CODE + 1589

: 2729 3265 1
: 2730 3266 1 END
: 2731 3267 0 ELUDOM

.EXTRN LIB\$SIGNAL, LIB\$STOP

PSECT SUMMARY

Name	Bytes	Attributes
\$OWNS	160	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
\$PLIT	508	NOVEC, NOWRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
\$GLOBAL\$	198	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
\$CODE	5559	NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	115	0	1000	00:02.1

COMMAND QUALIFIERS

; BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:ERF/OBJ=OBJ\$:ERF MSRC\$:ERF/UPDATE=(ENH\$:ERF)

: 2732 3268 0
: Size: 5559 code + 866 data bytes
: Run Time: 01:27.9
: Elapsed Time: 03:04.5
: Lines/CPU Min: 2230

ERF
V04-000

Errorlog Report Formatter

M 9
15-Sep-1984 23:42:14

VAX-11 Bliss-32 V4.0-742

Page 100

: Lexemes/CPU-Min: 15987
: Memory Used: 356 pages
: Compilation Complete

**

ERF1	ERF2	ERF3	ERF4	ERF5	ERF6	ERF7	ERF8	ERF9	ERF10	ERF11	ERF12	ERF13	ERF14	ERF15	ERF16	ERF17	ERF18	ERF19	ERF20	ERF21	ERF22	ERF23	ERF24	ERF25	ERF26	ERF27	ERF28	ERF29	ERF30	ERF31	ERF32	ERF33	ERF34	ERF35	ERF36	ERF37	ERF38	ERF39	ERF40	ERF41	ERF42	ERF43	ERF44	ERF45	ERF46	ERF47	ERF48	ERF49	ERF50	ERF51	ERF52	ERF53	ERF54	ERF55	ERF56	ERF57	ERF58	ERF59	ERF60	ERF61	ERF62	ERF63	ERF64	ERF65	ERF66	ERF67	ERF68	ERF69	ERF70	ERF71	ERF72	ERF73	ERF74	ERF75	ERF76	ERF77	ERF78	ERF79	ERF80	ERF81	ERF82	ERF83	ERF84	ERF85	ERF86	ERF87	ERF88	ERF89	ERF90	ERF91	ERF92	ERF93	ERF94	ERF95	ERF96	ERF97	ERF98	ERF99	ERF100
ERF101	ERF102	ERF103	ERF104	ERF105	ERF106	ERF107	ERF108	ERF109	ERF110	ERF111	ERF112	ERF113	ERF114	ERF115	ERF116	ERF117	ERF118	ERF119	ERF120	ERF121	ERF122	ERF123	ERF124	ERF125	ERF126	ERF127	ERF128	ERF129	ERF130	ERF131	ERF132	ERF133	ERF134	ERF135	ERF136	ERF137	ERF138	ERF139	ERF140	ERF141	ERF142	ERF143	ERF144	ERF145	ERF146	ERF147	ERF148	ERF149	ERF150	ERF151	ERF152	ERF153	ERF154	ERF155	ERF156	ERF157	ERF158	ERF159	ERF160	ERF161	ERF162	ERF163	ERF164	ERF165	ERF166	ERF167	ERF168	ERF169	ERF170	ERF171	ERF172	ERF173	ERF174	ERF175	ERF176	ERF177	ERF178	ERF179	ERF180	ERF181	ERF182	ERF183	ERF184	ERF185	ERF186	ERF187	ERF188	ERF189	ERF190	ERF191	ERF192	ERF193	ERF194	ERF195	ERF196	ERF197	ERF198	ERF199	ERF200
ERF201	ERF202	ERF203	ERF204	ERF205	ERF206	ERF207	ERF208	ERF209	ERF210	ERF211	ERF212	ERF213	ERF214	ERF215	ERF216	ERF217	ERF218	ERF219	ERF220	ERF221	ERF222	ERF223	ERF224	ERF225	ERF226	ERF227	ERF228	ERF229	ERF230	ERF231	ERF232	ERF233	ERF234	ERF235	ERF236	ERF237	ERF238	ERF239	ERF240	ERF241	ERF242	ERF243	ERF244	ERF245	ERF246	ERF247	ERF248	ERF249	ERF250	ERF251	ERF252	ERF253	ERF254	ERF255	ERF256	ERF257	ERF258	ERF259	ERF260	ERF261	ERF262	ERF263	ERF264	ERF265	ERF266	ERF267	ERF268	ERF269	ERF270	ERF271	ERF272	ERF273	ERF274	ERF275	ERF276	ERF277	ERF278	ERF279	ERF280	ERF281	ERF282	ERF283	ERF284	ERF285	ERF286	ERF287	ERF288	ERF289	ERF290	ERF291	ERF292	ERF293	ERF294	ERF295	ERF296	ERF297	ERF298	ERF299	ERF300
ERF301	ERF302	ERF303	ERF304	ERF305	ERF306	ERF307	ERF308	ERF309	ERF310	ERF311	ERF312	ERF313	ERF314	ERF315	ERF316	ERF317	ERF318	ERF319	ERF320	ERF321	ERF322	ERF323	ERF324	ERF325	ERF326	ERF327	ERF328	ERF329	ERF330	ERF331	ERF332	ERF333	ERF334	ERF335	ERF336	ERF337	ERF338	ERF339	ERF340	ERF341	ERF342	ERF343	ERF344	ERF345	ERF346	ERF347	ERF348	ERF349	ERF350	ERF351	ERF352	ERF353	ERF354	ERF355	ERF356	ERF357	ERF358	ERF359	ERF360	ERF361	ERF362	ERF363	ERF364	ERF365	ERF366	ERF367	ERF368	ERF369	ERF370	ERF371	ERF372	ERF373	ERF374	ERF375	ERF376	ERF377	ERF378	ERF379	ERF380	ERF381	ERF382	ERF383	ERF384	ERF385	ERF386	ERF387	ERF388	ERF389	ERF390	ERF391	ERF392	ERF393	ERF394	ERF395	ERF396	ERF397	ERF398	ERF399	ERF400
ERF401	ERF402	ERF403	ERF404	ERF405	ERF406	ERF407	ERF408	ERF409	ERF410	ERF411	ERF412	ERF413	ERF414	ERF415	ERF416	ERF417	ERF418	ERF419	ERF420	ERF421	ERF422	ERF423	ERF424	ERF425	ERF426	ERF427	ERF428	ERF429	ERF430	ERF431	ERF432	ERF433	ERF434	ERF435	ERF436	ERF437	ERF438	ERF439	ERF440	ERF441	ERF442	ERF443	ERF444	ERF445	ERF446	ERF447	ERF448	ERF449	ERF450	ERF451	ERF452	ERF453	ERF454	ERF455	ERF456	ERF457	ERF458	ERF459	ERF460	ERF461	ERF462	ERF463	ERF464	ERF465	ERF466	ERF467	ERF468	ERF469	ERF470	ERF471	ERF472	ERF473	ERF474	ERF475	ERF476	ERF477	ERF478	ERF479	ERF480	ERF481	ERF482	ERF483	ERF484	ERF485	ERF486	ERF487	ERF488	ERF489	ERF490	ERF491	ERF492	ERF493	ERF494	ERF495	ERF496	ERF497	ERF498	ERF499	ERF500
ERF501	ERF502	ERF503	ERF504	ERF505	ERF506	ERF507	ERF508	ERF509	ERF510	ERF511	ERF512	ERF513	ERF514	ERF515	ERF516	ERF517	ERF518	ERF519	ERF520	ERF521	ERF522	ERF523	ERF524	ERF525	ERF526	ERF527	ERF528	ERF529	ERF530	ERF531	ERF532	ERF533	ERF534	ERF535	ERF536	ERF537	ERF538	ERF539	ERF540	ERF541	ERF542	ERF543	ERF544	ERF545	ERF546	ERF547	ERF548	ERF549	ERF550	ERF551	ERF552	ERF553	ERF554	ERF555	ERF556	ERF557	ERF558	ERF559	ERF560	ERF561	ERF562	ERF563	ERF564	ERF565	ERF566	ERF567	ERF568	ERF569	ERF570	ERF571	ERF572	ERF573	ERF574	ERF575	ERF576	ERF577	ERF578	ERF579	ERF580	ERF581	ERF582	ERF583	ERF584	ERF585	ERF586	ERF587	ERF588	ERF589	ERF590	ERF591	ERF592	ERF593	ERF594	ERF595	ERF596	ERF597	ERF598	ERF599	ERF600
ERF601	ERF602	ERF603	ERF604	ERF605	ERF606	ERF607	ERF608	ERF609	ERF610	ERF611	ERF612	ERF613	ERF614	ERF615	ERF616	ERF617	ERF618	ERF619	ERF620	ERF621	ERF622	ERF623	ERF624	ERF625	ERF626	ERF627	ERF628	ERF629	ERF630	ERF631	ERF632	ERF633	ERF634	ERF635	ERF636	ERF637	ERF638	ERF639	ERF640	ERF641	ERF642	ERF643	ERF644	ERF645	ERF646	ERF647	ERF648	ERF649	ERF650	ERF651	ERF652	ERF653	ERF654	ERF655	ERF656	ERF657	ERF658	ERF659	ERF660	ERF661	ERF662	ERF663	ERF664	ERF665	ERF666	ERF667	ERF668	ERF669	ERF670	ERF671	ERF672	ERF673	ERF674	ERF675	ERF676	ERF677	ERF678	ERF679	ERF680	ERF681	ERF682	ERF683	ERF684	ERF685	ERF686	ERF687	ERF688	ERF689	ERF690	ERF691	ERF692	ERF693	ERF694	ERF695	ERF696	ERF697	ERF698	ERF699	ERF700
ERF701	ERF702	ERF703	ERF704	ERF705	ERF706	ERF707	ERF708	ERF709	ERF710	ERF711	ERF712	ERF713	ERF714	ERF715	ERF716	ERF717	ERF718	ERF719	ERF720	ERF721	ERF722	ERF723	ERF724	ERF725	ERF726	ERF727	ERF728	ERF729	ERF730	ERF731	ERF732	ERF733	ERF734	ERF735	ERF736	ERF737	ERF738	ERF739	ERF740	ERF741	ERF742	ERF743	ERF744	ERF745	ERF746	ERF747	ERF748	ERF749	ERF750	ERF751	ERF752	ERF753	ERF754	ERF755	ERF756	ERF757	ERF758	ERF759	ERF760	ERF761	ERF762	ERF763	ERF764	ERF765	ERF766	ERF767	ERF768	ERF769	ERF770	ERF771	ERF772	ERF773	ERF774	ERF775	ERF776	ERF777	ERF778	ERF779	ERF780	ERF781	ERF782	ERF783	ERF784	ERF785	ERF786	ERF787	ERF788	ERF789	ERF790	ERF791	ERF792	ERF793	ERF794	ERF795	ERF796	ERF797	ERF798	ERF799	ERF800
ERF801	ERF802	ERF803	ERF804	ERF805	ERF806	ERF807	ERF808	ERF809	ERF810	ERF811	ERF812	ERF813	ERF814	ERF815	ERF816	ERF817	ERF818	ERF819	ERF820	ERF821	ERF822	ERF823	ERF824	ERF825	ERF826	ERF827	ERF828	ERF829	ERF830	ERF831	ERF832	ERF833	ERF834	ERF835	ERF836	ERF837	ERF838	ERF839	ERF840	ERF841	ERF842	ERF843	ERF844	ERF845	ERF846	ERF847	ERF848	ERF849	ERF850	ERF851	ERF852	ERF853	ERF854	ERF855	ERF856	ERF857	ERF858	ERF859	ERF860	ERF861	ERF862	ERF863	ERF864	ERF865	ERF866	ERF867	ERF868	ERF869	ERF870	ERF871	ERF872	ERF873	ERF874	ERF875	ERF876	ERF877	ERF878	ERF879	ERF880	ERF881	ERF882	ERF883	ERF884	ERF885	ERF886	ERF887	ERF888	ERF889	ERF890	ERF891	ERF892	ERF893	ERF894	ERF895	ERF896	ERF897	ERF898	ERF899	ERF900
ERF901	ERF902	ERF903	ERF904	ERF905	ERF906	ERF907	ERF908	ERF909	ERF910	ERF911	ERF912	ERF913	ERF914	ERF915	ERF916	ERF917	ERF918	ERF919	ERF920	ERF921	ERF922	ERF923	ERF924	ERF925	ERF926	ERF927	ERF928	ERF929	ERF930	ERF931	ERF932	ERF933	ERF934	ERF935	ERF936	ERF937	ERF938	ERF939	ERF940	ERF941	ERF942	ERF943	ERF944	ERF945	ERF946	ERF947	ERF948	ERF949	ERF950	ERF951	ERF952	ERF953	ERF954	ERF955	ERF956	ERF957	ERF958	ERF959	ERF960	ERF961	ERF962	ERF963	ERF964	ERF965	ERF966	ERF967	ERF968	ERF969	ERF970	ERF971	ERF972	ERF973	ERF974	ERF975	ERF976	ERF977	ERF978	ERF979	ERF980	ERF981	ERF982	ERF983	ERF984	ERF985	ERF986	ERF987	ERF988	ERF989	ERF990	ERF991	ERF992	ERF993	ERF994	ERF995	ERF996	ERF997	ERF998	ERF999	ERF1000